SEGUIMIENTO DE UN OBJETIVO MOVÍL APLICANDO MONOVISIÓN EN EL SISTEMA DE NAVEGACIÓN ÁEREA AUTÓNOMA UTILIZANDO GPU

JAVIER ALEXIS ABDOR SIERRA
DANIEL FELIPE DELGADO VILLAFAÑE

UNIVERSIDAD PILOTO DE COLOMBIA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA MECATRÓNICA
BOGOTA D.C.
2017

SEGUIMIENTO DE UN OBJETIVO MOVÍL APLICANDO MONOVISIÓN EN EL SISTEMA DE NAVEGACIÓN ÁEREA AUTÓNOMA UTILIZANDO GPU

JAVIER ALEXIS ABDOR SIERRA
DANIEL FELIPE DELGADO VILLAFAÑE

MONOGRAFÍA DE TRABAJO DE GRADO PARA OPTAR AL TÍTULO DE INGENIERO MECATRÓNICO

DIRECTOR: ING. MSC. RUBEN DARIO HERNANDEZ BELEÑO

UNIVERSIDAD PILOTO DE COLOMBIA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA MECATRÓNICA
BOGOTA D.C.
2017

Nota de aceptación:

El trabajo de grado, titulado **"SEGUIMIENTO DE UN OBJETIVO MOVÍL APLICANDO MONOVISIÓN EN EL SISTEMA DE NAVEGACIÓN ÁEREA AUTÓNOMA UTILIZANDO GPU"** elaborado y presentado por los estudiantes Javier Abdor y Daniel Delgado, como requisito parcial para optar al título de Ingeniero Mecatrónico, fue aprobado por el Jurado Calificador.

Director del proyecto

DEDICATORIA

Dedico este trabajo principalmente a mi madre por guiarme y formarme con buenos hábitos y valores, ya que gracias a ella soy el ser humano integro que soy hoy en día. A mi padre y abuela quienes me apoyaron en todas las oportunidades que se presentaron durante mi formación académica.

Javier Abdor

Este trabajo quisiera dedicárselo a mis padres por ser el pilar fundamental en todo lo que soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo en cada proyecto que me he propuesto. Gracias por la persona que hoy en día soy.

Daniel Delgado

AGRACEDEMIENTO

Agradezco a mi familia por su apoyo incondicional y económico. A mi director el Ingeniero MSc. Rubén Darío Hernández Beleño por compartir su conocimiento, su paciencia, creer en nosotros y más que todo ser una gran persona. A mi novia por su apoyo incondicional en los momentos en que las situaciones eran de dificultad. A mis amigos cercanos por brindarnos ayuda en los momentos en que la necesitamos y a la Universidad Piloto de Colombia por desarrollar un espacio apto para mi formación profesional y personal.

Javier Abdor

Le agradezco a Dios por haberme acompañado y guiado a lo largo de mi carrera, por ser mi fortaleza en los momentos de debilidad y por brindarme una vida llena de aprendizajes y experiencias. Le doy gracias a mis padres Omar y Monica por apoyarme en todo momento, por los valores que me han inculcado, por haberme dado la oportunidad de tener una excelente educación en el trascurso de mi vida y por ese ejemplo de personas a la que algún día quisiera ser. A mis hermanos Omar Andres y Camila por estar siempre conmigo aun así estando lejos, también por ser en muchos aspectos un ejemplo en mi vida. Gracias al Ing. M.Sc. Ruben Dario Hernández por creer en nosotros, y habernos brindado la oportunidad de desarrollar nuestro proyecto de grado en conjunto con usted, por compartir su conocimiento y sobre todo su amistad. A Javier por haber sido un excelente compañero y amigo, por haberme tenido paciencia en todos los momentos complicados que pasamos durante todo nuestro paso por la universidad y sobre todo por ser una excelente persona. A Camila por todo el apoyo brindado, por darnos muchas alegrías durante nuestros últimos semestres, por enseñarme las cosas que valen la pena en la vida y sobre todo por permitirme conocer a su bella familia.

Daniel Delgado

Índice

Índice de figuras

Índice de tablas

RESUMEN

El presente documento describe el desarrollo e implementación de controladores difusos y PID a un quadrotor tipo comercial. Este es capaz de seguir un objeto por medio de visión por computador calculando el centroide del objeto generando así una serie de coordenadas las cuales se compararán con la posición actual del vehículo y la deseada, para dar así los errores en cada una de sus variables. El control descrito tiene como entrada los errores en cada uno de sus cuatro variables (pitch, roll, yaw y altura), que son otorgados gracias a los sensores que el cuadrotor contiene. Para la validación del modelado dinámico y cinemático del quadrotor se realizaron una serie de simulaciones para que el quadrotor siguiera una serie de trayectorias dejando ver el comportamiento de cada una de sus cuatro variables.

En relación con la visión por computador se desarrolla un algoritmo de procesamiento de imágenes en serie que tiene como objetivo la extracción del objeto por su color. Ya con la extracción del objeto se calcula el centroide del objeto para el punto de referencia del controlador. Adicionalmente se propone un algoritmo de procesamiento de imágenes en paralelo utilizando CUDA que tiene como objetivo la segmentación por medio de la transformada watershed. Este algoritmo tiene un pre-proceso compuesto por un filtro gaussiano, un filtro sobel y una reconstrucción morfológica en escala de grises. En este documento se describe los filtros mencionados anteriormente y la implementación de los filtros gaussiano y sobel en paralelo ya que la reconstrucción morfológica y transformada watershed se proponen para un trabajo futuro.

Palabras claves - difuso, UAV, visión por computador, simulación, comportamiento, procesamiento de imágenes, paralelo, transformada watershed.

ABSTRACT

This document describes the development and implementation of a fuzzy controller applied to a commercial aerial vehicle (UAV). This vehicle will be able to follow an object by utilizing computer vision and calculating the centroid of the object thus generating a series of coordinates which will be compared with the current position of the vehicle and the desired one, therefore giving the errors in each of its variables. The control described has as input the errors in each of its four variables (pitch, roll, yaw and altitude), which are gathered thanks to the sensors that the vehicle contains. As for the validation of the dynamic and kinematic model of the vehicle, a series of simulations were carried out to allow the vehicle to follow a series of trajectories to verify its behavior in its four variables.

On the other hand, a serial image processing algorithm was developed for the extraction of the object by its color. Once the object is extracted the centroid of the object is calculated and is the reference for the controller. In addition, a parallel image processing algorithm is proposed for image segmentation using the watershed transform. This algorithm has a preprocess where a Gaussian filter, Sobel filter and morphological reconstruction in grayscale is performed before applying the watershed transform. In this document the filters will be briefly discussed and the implementation of the Gaussian filter and Sobel filter in parallel will be explained since the morphological reconstruction and watershed transform will be proposed for future work.

Keywords - fuzzy, UAV, computer vision, simulation, behavior, image processing, parallel, watershed transform.

1. Introducción

El desarrollo tecnológico alrededor de los UAVs (Unmanned Aerial Vehicle) o vehículos aéreos no tripulados, es uno de los de mayor evolución en los últimos años. A pesar de que estos dispositivos en sus inicios, fueron concebidos con fines militares han tenido gran acogida en aplicaciones comerciales. Como lo describe William Pinilla, docente de la Escuela de Suboficiales de la Fuerza Aérea Colombiana, quien menciona que "esta tecnología fue pensada inicialmente como un aeronave robot, capaz de volar de forma autónoma y cumplir una misión con unos objetivos específicos y definidos"[1].

A su vez, la vinculación de estrategias robóticas, como lo son sistemas de visión computacional, son como lo menciona Kasturi. J "la suma de aspecto geométricos, de medición e interpretación de imágenes"[2]. Por otro lado, se ha visto el incremento de las aplicaciones en estos dispositivos autónomos. Algunos ejemplos de las mencionadas aplicaciones son, el evitar obstáculos, hacer un mapeo de zonas desconocidas, tareas de posicionamiento y seguimiento de objetivos. En este sentido, el uso de los sistemas de visión es apropiado y prometedor debido a su facilidad para obtener información del entorno, por lo que la visión computacional es un elemento clave en el proceso de automatización de cualquier tipo de vehículo, más aun si se pretende llevar a cabo una misión de seguimiento de objetos en movimiento. Por lo que se ha generado un gran interés en campos investigativos como la robótica, ya que al experimentar cambios lumínicos en entornos reales, genera cierto grado de complejidad.

Por esto mismo los investigadores están recurriendo al estudio de nuevas técnicas para procesar y optimizar el rendimiento de sus aplicaciones, una de las más usadas es el procesamiento de imágenes, más conocido como visión artificial. Este ha dejado a un lado el trabajo con sensores. El campo de visión artificial ha sido muy estudiado, tanto así que grandes empresas como lo es NVidia se han dado a la tarea de generar tarjetas gráficas las cuales permiten que el usuario pueda programar. Al mismo tiempo estas tarjetas cuentan con su propia arquitectura de programación llamada CUDA, la cual como se mencionará los beneficios que le proporciona más adelante optimiza las aplicaciones.

1.1. Planteamiento y Formulación del Problema

1.1.1. Descripción del problema

En relación con los avances tecnológicos en vehículos aéreos no tripulados, se ha visto el surgimiento de múltiples aplicaciones para estos mismos; en relación más específicamente con el robot aéreo tipo quadrotor. Algunas de estas aplicaciones se

evidencian en los sectores de agricultura, vigilancia autónoma y el campo militar. Así mismo, se ha usado como plataforma de investigación en varias universidades del mundo, ya que sus variadas aplicaciones proponen un nuevo sistema que facilita la independencia de estos mismos a la manipulación humana.

Por otro lado, el desarrollo de estas aplicaciones en el campo, tiene una notada falencia en cuanto al uso de la arquitectura CUDA, así como la programación en paralelo en quadrotores. La importancia radica en que la aplicación facilita la programación en tarjetas gráficas NVIDIA, lo que acelera procesos de procesamiento de imágenes y entrenamiento de redes neuronales, que a su vez es una gran herramienta para disminuir costos de tiempo y esfuerzo.

Por lo anterior, lo que se busca con esta investigación es desarrollar y aplicar un algoritmo de procesamiento de imágenes con librerías OpenCV y proponer un algoritmo con arquitectura CUDA para una plataforma aérea AR Drone para el reconocimiento y seguimiento de objetos de forma autónoma, lo que dará una nueva perspectiva en relación a la robótica cooperativa.

1.1.2. Formulación del problema

¿Cómo desarrollar e implementar un algoritmo de procesamiento de imágenes usando las librerías OpenCV y arquitectura CUDA a un vehículo aéreo tipo quadrotor para detección y seguimiento de objetos de forma autónoma?

1.2. Justificación

Viendo las múltiples aplicaciones que se realizan con dispositivos aéreos, más directamente con quadrotores tipo AR Drone, lo que se busca es el desarrollo y mejoramiento de el control y percepción, ya que esto dará paso a la implementación de un seguimiento a un objeto por medio de arquitectura CUDA y ayudará a su vez, al progreso en varios campos investigativos. Por otro lado, este trabajo servirá de apoyo material para el desarrollo del algoritmo propuesto en paralelo o trabajos futuros en arquitectura CUDA para los estudiantes de la Universidad Piloto de Colombia.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar y aplicar un algoritmo de procesamiento de imágenes a un robot aéreo (AR Drone) para el reconocimiento y seguimiento de objetos en ambientes

externos.

1.3.2. Objetivos específicos

- Analizar el modelo matemático utilizado para un robot aéreo tipo quadrotor.
- Implementar y simular el modelo obtenido en Matlab®.
- Implementar un algoritmo de procesamiento de imágenes en el lenguaje CUDA en paralelo o con las librerías OpenCV en serie.
- Realizar un sistema de control el cual permita la actuación del sistema para el seguimiento en ambientes externos.
- Validar el comportamiento del robot aéreo en el seguimiento de un objetivo móvil en un ambiente controlado.

1.4. Alcances y Limitaciones

La propuesta se plantea desarrollar en un ambiente controlado, de esta forma se usará el robot aéreo AR Drone, el cual brinda un control en precisión y una estabilidad automática al momento de su ejecución ya que cuenta con distintos sensores como un giroscopio, magnetómetro, sensor de presión y ultrasonido. Este también incorpora dos cámaras de diferente resolución la cual transmitirá la información a un servidor y posteriormente se hará su respectivo procesamiento de imagen, así, permitiendo generar proyectos futuros en el desarrollo de drones.

Para el procesamiento de imágenes se utiliza la librería OpenCV para el algoritmo en serie y se tendrá en cuenta el rendimiento. Más aun, para el algoritmo de procesamiento de imágenes en paralelo propuesto se hace uso solamente de la arquitectura CUDA.

Dado que este proyecto se llevara a cabo en un ambiente controlado, se tendrá en cuenta algunos parámetros, entre ellos la distancia de comunicación del AR Drone con el dispositivo conectado, la cual es de 50 metros, si esta es mayor a dicha distancia puede causar inestabilidad en la comunicación con el robot aéreo y afecte la calidad de la señal de las cámaras. Otro aspecto visto en este dispositivo es el tiempo de vuelo, dado que cuenta con una batería de litio de 1000 mAh, la cual da un tiempo de ejecución de aproximadamente 12 minutos.

Por otro lado, es importante mencionar que el algoritmo en arquitectura CUDA solo se puede ejecutar en GPUs de la empresa NVidia lo cual no se podrán usar tarjetas gráficas de otros fabricantes como MSI y Gigabyte. También los más de 2500

algoritmos que se pueden implementar con la librería OpenCV, con la arquitectura CUDA ya que todavía está en desarrollo se pueden implementar más de 500 algoritmos para procesamiento de imágenes en la GPU. Adicionalmente la iluminación en el ambiente exterior contiene un efecto al momento de procesar las imágenes y genera restricciones en algunas aplicaciones.

1.5. Línea de Investigación del Programa

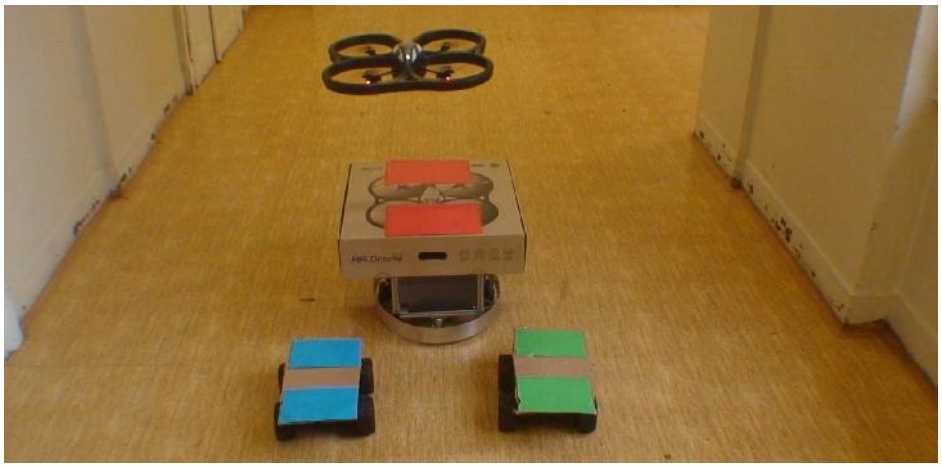
El desarrollo del proyecto tiene como enfoque implementar un algoritmo de procesamiento de imágenes para el reconocimiento y seguimiento de un objeto con un quadrotor (AR Drone). Dicho algoritmo se implementara en serie utilizando las librerías OpenCV y por la parte de procesamiento en paralelo se realizara en una tarjeta GPU de NVIDIA GeForce GT 540M con arquitectura CUDA. Dado lo anterior, se establece como línea de investigación del proyecto la robótica y biomecatrónica.

2. Estado del Arte

Al paso del tiempo los vehículos aéreos tipo quadrotor han pasado de ser un simple dispositivo de ocio a ser utilizados en el campo de la robótica, para diferentes aplicaciones o resolver problemas de la vida cotidiana. El AR Drone 2.0, por su bajo costo es una de las plataformas más utilizadas para la investigación.

Recientemente en la Universidad Técnica Checa en Praga se han desarrollaron dos (2) proyectos con el AR Drone 2.0 dando uso a todos sus sensores y cámaras. El primero de estos, es un sistema de ubicación externo, el cual tiene como finalidad probar un método de control de formación, que permite cambiar adaptativamente la formación, de acuerdo con las limitaciones del entorno. El funcionamiento es por medio del uso de la cámara de la parte inferior del AR Drone, para determinar la posición de los objetos por sus distintos colores. El método de enfoque es de líder-seguidor, en el cual el robot líder planea y ejecuta la trayectoria, mientras los robots seguidores mantienen una posición relativa a la del líder como se puede observar en la figura 1.

Figura 1: Sistema de ubicación externo



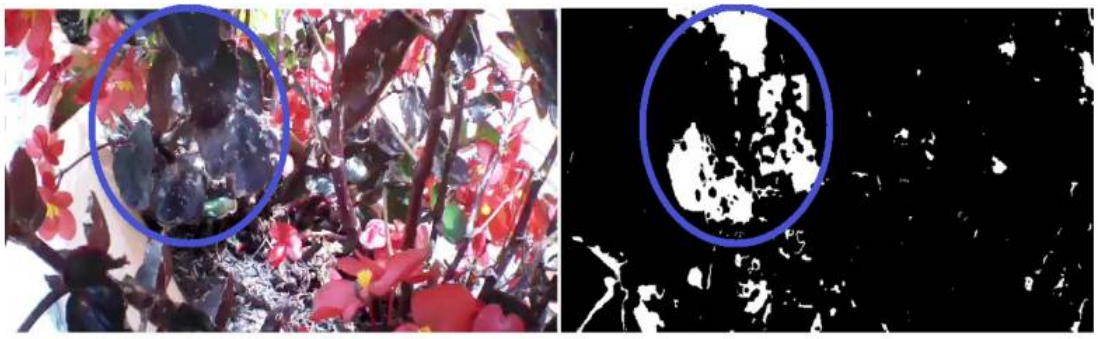
Fuente: KRANIK

Sin embargo, los robots seguidores se pueden separar del líder por la presencia de obstáculos en el entorno, lo que causa el desequilibrio de su posicionamiento frente al líder. A su vez, en este proceso se emplea el AR Drone, el cual vuela sobre el robot líder y le proporciona a los seguidores unas posiciones establecidas, en relación al líder. Ya con una posición determinada, los seguidores ajustan su velocidad para mantener la formación deseada[3].

Por otra parte, la utilización de los drones trasciende al sector agrónomo; con base

en los beneficios que este proporciona, se da la necesidad de desarrollar nuevas aplicaciones para estos. Ejemplo de esto, es la realizada en Colombia por la Universidad Militar Nueva Granada, la cual incorpora un algoritmo de procesamiento de imágenes a un quadrotor para que reconociera los daños causados por plagas a un cultivo de Begonias. Para esto fue necesario la utilización de filtros morfológicos, difuminación gaussiano y filtrado HSL; este algoritmo de reconocimiento de patrones se enfoca en los agujeros de las hojas de las plantas[4].

Figura 2: Reconocimiento de daños causados por plagas



Fuente: CACERES

Otro proyecto de investigación con el AR Drone, fue desarrollado en la Universidad Técnica Checa en Praga, el cual se desarrolló en base a la vigilancia autónoma, es decir, que el AR Drone vigila diferentes objetos de interés. Este vuela de forma autónoma en la parte superior de los objetos de interés y captura las imágenes lo más frecuente posible.

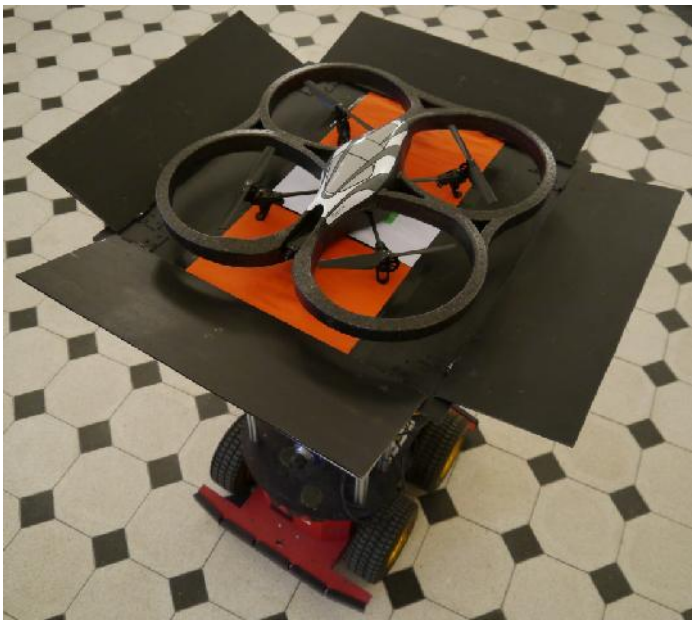
Figura 3: AR Drone tomando imagen del objeto

Fuente: KRANIK

Para hacer esto posible, emplearon un método de navegación, en el cual se establece el orden que el AR Drone debe tomar para ir a cada objeto de interés que se puede evidenciar en la figura 3.

Así mismo, en la Universidad Checa, se empleó otro sistema de vigilancia autónomo con un UGV(Vehículo Terrestre no Tripulado) y el AR Drone 2.0 para ambientes internos. El UGV tiene un helipuerto interactivo en la parte superior lo cual es el portador del AR Drone como se muestra en la figura 4.

Figura 4: UGV con AR Drone



Fuente: SASKA

En esta investigación los aspectos importantes en los cuales se enfocaron, son la navegación visual, la ubicación y el aterrizaje que se tiene que hacer periódicamente. El objetivo del UGV es seguir una trayectoria establecida, tener una fuente de energía de alta capacidad y servir como portador del AR Drone; en el momento que el UGV se encuentre con un obstáculo el AR Drone despega automáticamente y resume la inspección usando las dos cámaras; ya que se conoce la última posición del UGV en el momento de despegar, se usa un método de localización para determinar la ubicación absoluta del UGV. Ya capturadas las imágenes de interés el AR Drone regresa al UGV y sigue con la trayectoria planeada[5].

Otra aplicación en la robótica cooperativa es el AR Drone 2.0 con el robot llamado LAURON IV; este robot hexápodo inspirado por el insecto palo tiene tres grados de libertad en cada pata y dos en su cabeza, por lo que equivale un total de 20 gra-

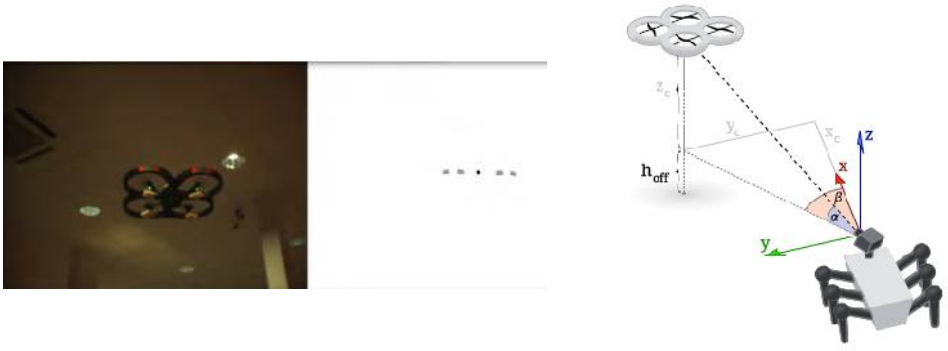
dos de libertad. Su amplia disponibilidad de sensores, LAURON es una plataforma para exploración, búsqueda y rescate, e inspección; además navega por terrenos ásperos, no estructurados y áreas que no son seguras para humanos[6].

Figura 5: AR Drone con LAURON IV

Fuente: HEPPNER

Para este proyecto se quiere hacer uso del AR Drone 2.0 con el fin de inspeccionar áreas de interés antes que LAURON se dirija a este lugar. En la figura 5 muestra la vista de LAURON hacia el AR Drone 2.0 y la forma que calcula la distancia de cada uno de los robots.

Figura 6: Visión del robot LAURON hacia el AR Drone



Fuente: HEPPNER

Como el AR Drone tiene un tiempo estimado de 12 minutos de vuelo, LAURON

tiene un punto de aterrizaje en la parte superior y no tendrá un efecto negativo, ya que solo agrega 420g de peso adicional. En general los robots tuvieron un rendimiento bueno aunque tuviera un error de posición de 4 cm a 80 cm.

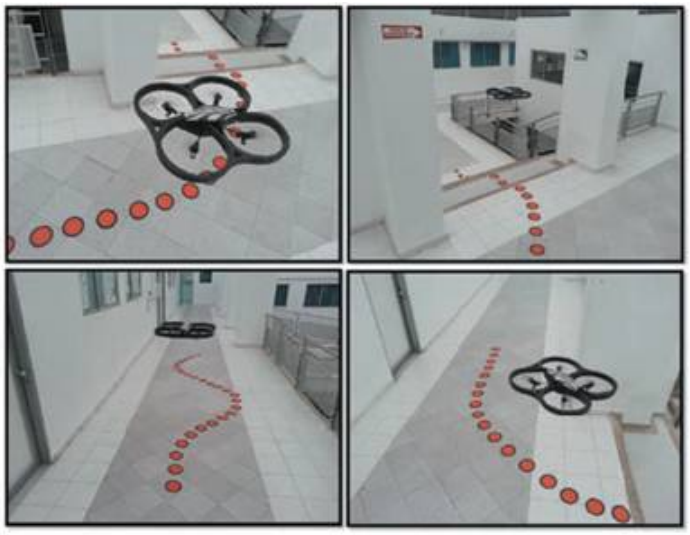
Por otro lado en la Universidad de Catania en Italia se implementó un quadrotor y un robot terrestre para planear rutas de navegación en terrenos irregulares con minas explosivas. El objetivo de esta implementación, es el uso del quadrotor en el seguimiento del robot terrestre de forma autónoma usando un algoritmo de procesamiento de imágenes; de esta forma el operador solo controla el robot terrestre mientras el quadrotor vuela sobre el área de operación.

Figura 7: Sistema de posición del quadrotor

Fuente: CANTELLI

El algoritmo de procesamiento de imágenes se realizó en ocho pasos, la adquisición de imagen y umbralización, extracción de pixeles blancos de la imagen original, extracción de pixeles negros para identificar el chasis del quadrotor, un lógico AND para identificar los LEDs blancos en las esquinas del chasis, validación geométrica de los puntos detectados la cual debe ser un cuadrado, estimación de posición de pixeles al espacio euclidiano y por último, aplicar técnicas de filtro Kalman para reducir ruido del ambiente y mejorar la ubicación cuando el quadrotor no es reconocido por el algoritmo. Este sistema fue empleado para operaciones de humanitario antiminas para el proyecto TIRAMISU[7].

Figura 8: Pruebas de seguimiento de trayectoria



Fuente: SILVA

Los drones son comunes en los sistemas de vigilancia, lo que permite la ampliación de aplicaciones para estos vehículos aéreos. En la Universidad de Pamplona en Colombia, se desarrolló un sistema de inspección y vigilancia, el cual tenía como objetivo el procesamiento de imágenes dentro de las cámaras del robot, con el fin de realizar una navegación autónoma o semiautónoma. La estrategia de control implementada para este dispositivo fue un control servo-visual, el cual determinaba la orientación y la posición del mismo, para así seguir una trayectoria demarcada.

Para verificar los controladores realizaron diversas pruebas en las que incluyen el posicionamiento del robot sobre un patrón (control de posición), el cambio de orientación usando al menos dos patrones (control de orientación) y el seguimiento de trayectorias completas, dando así resultados satisfactorios a pesar de las perturbaciones experimentadas, como el viento y los cambios de iluminación[8].

Mas aún, el proyecto desarrollado en la Universidad Estatual de Campinas se basa en un control servo visual en vehículos robóticos terrestres, quien lleva un seguimiento y búsqueda de un objetivo. La selección de este se basa en la coincidencia de la región diana de la secuencia de imágenes, y la persecución por el movimiento de la navegación basada en la generación de la información de la zona de destino.

Para el procesamiento de detección, se utilizó estrategias en procesamiento de imágenes como el filtro gaussiano KNN, el filtro Sobel y reconstrucción morfológica H-MIN. Por último, se aplica la transformada watershed para la segmentación de objetos en la imagen como se muestra en la figura 9. Después de aplicar la transfor-

mada watershed se busca los contornos y el centroide de cada parte segmentada para ya tener cada característica de la región segmentada. Aunque, esta técnica no es compatible en sistemas en tiempo real, se utilizó esta para moderar los mencionados algoritmos para el procesamiento en paralelo con arquitectura CUDA[9].

Figura 9: Imagen resultante después de la aplicación de la transformada watershed

Fuente: BERNARDES

En relación con lo anterior, un estudio desarrollado en la Universidad del Cabo Occidental y Universidad Rhodes en Sudáfrica, en el que se compara la velocidad de procesamiento de imágenes con la CPU y la GPU. Se aplicó un conversión de color y filtro de suavizado gaussiano, a una imagen de 1280x1024 y tuvo un mejoramiento de 2.7 veces más en la GPU comparado con la CPU. Otra prueba hecha fue aplicar detección de bordes a varias imágenes de diferentes tamaños, la figura 10 muestra los resultados de las pruebas.

Figura 10: Comparacion de tiempos de procesamiento

Tamaño Imagen	CUDA (ms)	OpenCV (ms)	Mejoramiento
256 x 256	1.82	3.06	1.68
512 x 512	3.40	8.28	2.44
1024 x 1024	10.92	28.92	2.65
2048 x 2048	31.46	105.55	3.35
3936 x 3936	96.62	374.36	3.87

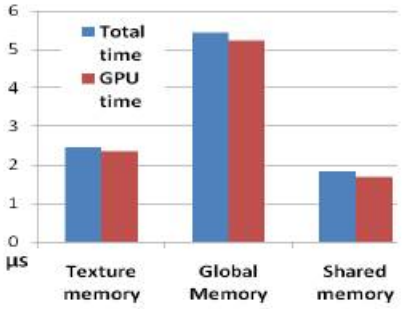
Fuente: BROWN

Como se aprecia en la tabla, la arquitectura CUDA aumenta el rendimiento signi-

ficativamente del algoritmo. Con base estos resultados, se implementó CUDA para detección de posturas del cuerpo superior para traducción automática de South African Sign Language (Lenguaje de señas de Sudáfrica) a inglés[10].

Por otro lado, otro estudio realizado en el Instituto Politécnico y Universidad Estatal de Virginia se implementó el algoritmo de Canny (detección de bordes) y el de detección de movimiento en video conocido como “Vector Coherence Mapping” (VCM). Para el algoritmo de Canny se utilizó tres diferentes regiones de memoria y patrones de exceso; las regiones de memoria son “texture memory”, memoria global y memoria compartida. De estas tres regiones de memorias, el que obtuvo mejor desempeño, fue la memoria compartida, como lo muestra la tabla a continuación.

Figura 11: Tiempo de procesamiento en diferentes memorias



	Total	GPU time	%GPU time
Texture memory	2.459	2.344	95.34
Global Memory	5.433	5.225	96.17
Shared memory	1.818	1.706	93.87

Fuente: YONG

Para el algoritmo VCM, el cual es el más intensivo computacionalmente, requiere de un proceso mucho más rápido para detectar el movimiento. Aplicando la arquitectura CUDA tuvo un mejoramiento de 22 veces mejor que con la CPU[11].

Adicional a ello, en la Universidad de Massachusetts Lowell, se aplicó CUDA para el algoritmo basado en filtrado de partículas SLAM (Localización y Mapeo Simultáneos) con el fin de mejorar su desempeño. En la robótica este algoritmo tiene muchas aplicaciones pero tiene un costo computacional alto, ya que las computaciones de estas partículas son independientes de cada una en este algoritmo. El cálculo del peso de las partículas es el que más tiempo consume y con el procesamiento en CUDA se obtuvo un mejor desempeño de una magnitud o más de ellas[12].

Por otro lado, se hizo un estudio para implementar el escaneo laser para visión robótica usando CUDA; en este se usó un láser que envía un pulso de luz y mide

el tiempo que tarda en regresar después de pegar en un objeto. En esta aplicación el láser se usa para reconocimiento de objetos en un robot. Se tiene un láser que envía el pulso de luz y espera 'que regrese, después sigue a la línea posterior hasta que termine todo el barrido. Hasta el momento la aplicación completa 1000 barridos y procesar todos los datos en un tiempo razonable es una tarea de difícil ejecución. Los resultados de esta investigación fueron positivos, ya que mostro un mejoramiento en el proceso de datos como lo revela la figura 12.

Figura 12: CUDA vs. Programación secuencial

Líneas Escaneadas	Paralelo (CUDA) (s)	Secuencial (s)
640	0.0054	1.448
1024	0,00545	`2.289
2500	0.00573	5.569
5000	0.00655	11.12
10000	0.00566	22.634

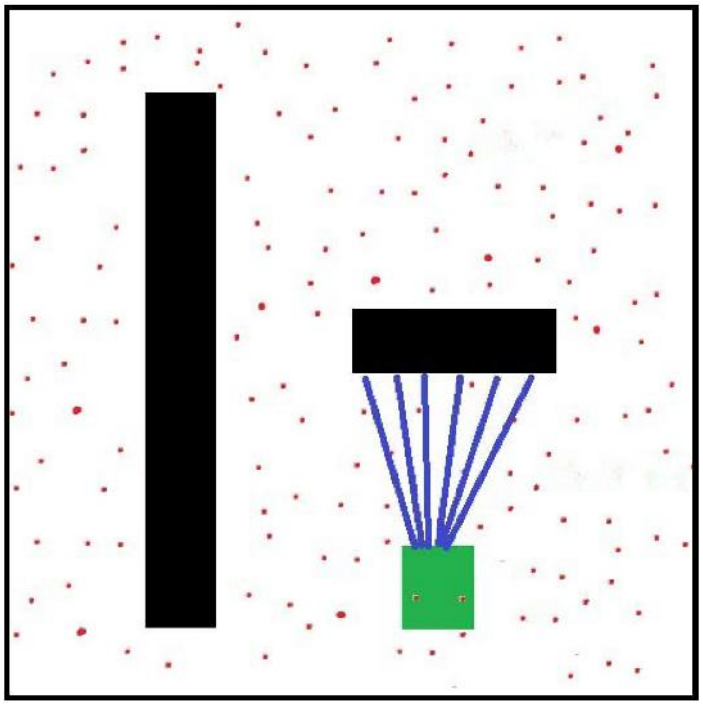
Fuente: SHINSEL

La importancia de estos resultados también es parte fundamental en otras aplicaciones. Por ejemplo, cuando el escaneo de láser es usado para detección de objetos en un vehículo en movimiento, el tiempo que toma en procesar los datos es crítico para que el sistema responda en un tiempo adecuado[13].

Por otra parte, en la Universidad Tomsk Polytechnic en Rusia se implementó la localización basada en partículas del algoritmo Monte - Carlo en un robot de tres ruedas. Este algoritmo tiene muchas aplicaciones en la robótica pero requiere alto nivel computacional. Para el desarrollo del proyecto se implementó en el sensor Microsoft Kinect y odometría robótica para hacer localización en un ambiente interno; a su vez, se usó la tarjeta gráfica de NVIDIA Jetson TK1 para sistemas embebidos ya que por su tamaño pequeño se puede implementar directamente en el robot. Esta tarjeta cuenta con 192 núcleos cuda y el sistema operativo Linux lo que hace posible la integración del Microsoft Kinect. Hay seis pasos principales para implementar el algoritmo Monte - Carlo; primero se genera un conjunto de partículas y se inician con una posición y peso al alzar, segundo se aplica el modelo de movimiento del robot a las partículas para predecir el estado siguiente del robot, tercero se siente el entorno y se compara las mediciones actuales con las predecidas, cuarto se calcula el peso de las partículas dependiendo de qué tan bien cada partícula predijo el estado actual del robot, quinto se vuelve a muestrear la distribución y la probabilidad de elegir a cada partícula es proporcional a su eso y por último se vuelve al paso dos para repetir el ciclo. La siguiente figura muestra una el modelo útil sensorial para el

algoritmo Monte-Carlo.

Figura 13: Posición del robot con visión del Kinect

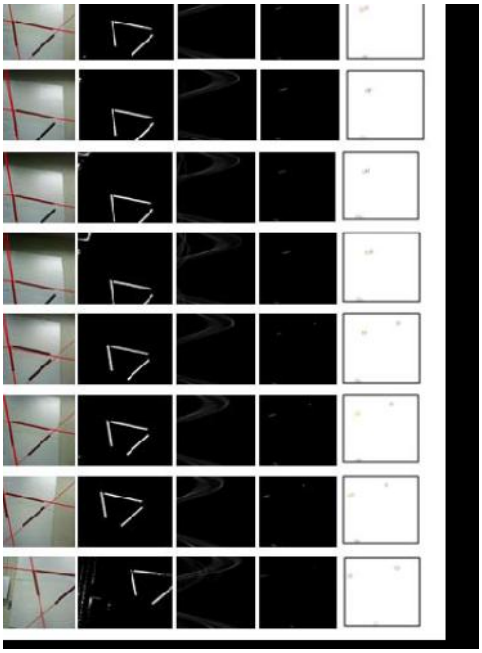


Fuente: RUD

Este algoritmo se probó con 512 partículas hasta 262.144 para tener estadísticas precisas. Los resultados de esta investigación muestran un desempeño diez veces mejor con la arquitectura CUDA que con procesamiento con la CPU. Se evidencia un desempeño más eficaz en el peso de partículas con 15 veces mejor desempeño[14].

En otro estudio reciente en la Universidad Konkuk en la República de Corea, se aplicó un reconocimiento de multilíneas en tiempo real por visión computacional para un UAV (Vehículo Aéreo no tripulado). Se observó que muchas pistas de aterrizaje tienen líneas, lo que facilita el enfoque en el rastreo de líneas para un aterrizaje autónomo. Las técnicas aplicadas para el procesamiento de multilíneas, son la transformación de Hough y el filtro de Kalman. La transformación de Hough es usada para la extracción de líneas y el filtro de Kalman predice estados futuros. La transformación de Hough es fácil de implementar y es robusto hacia el ruido, pero el consumo de recursos crece exponencialmente por la resolución de la imagen o por el requerimiento de la alta precisión en el espacio de Hough; por ende, se implementa el algoritmo en una GPU con la arquitectura CUDA; en la siguiente figura se puede observar el proceso del algoritmo.

Figura 14: Procedimiento de rastreo de líneas

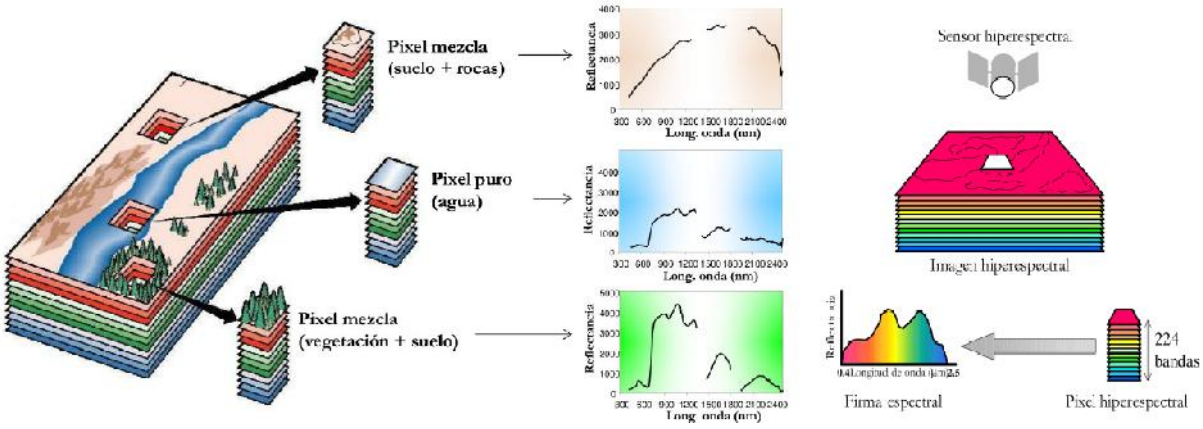


Fuente: VLADIMIR

EL objetivo inicial de mejorar el rendimiento del algoritmo con arquitectura CUDA, se realizó satisfactoriamente. Ejemplo de esto, es el hecho de que el algoritmo de video tuvo 110 frames por segundo, en comparación a lo obtenido con la CPU que fueron 20 de estas; lo cual indica su validez en un UAV real[15].

Así mismo la arquitectura CUDA se ha adentrado en aplicaciones con tecnología hiperespectral, las cuales se enfocan en la observación remota de la tierra. Todo esto, da como resultado el desarrollo de instrumentos de medida de muy alta resolución en dominios espacial y espectral. Basado en lo anterior, en la Universidad de Extremadura España, se implementó un algoritmo que se encargaba de la extracción de referencias espectrales puras en imágenes hiperespectrales de la superficie terrestre, haciendo uso de tarjetas gráficas programables GPUs de NVDIA para acelerar los cómputos relativos al proceso. Para esto se realizó un estudio experimental sobre la precisión y rendimiento en paralelo de la implementación desarrollada utilizando imágenes hiperespectrales reales adquiridas por el sensor Airbone Visible InfraRed Imaging Spectrometrer (AVIRIS) de NASA Jet Propulsion Laboratory[16].

Figura 15: Procedimiento de análisis Hiperespectral



Fuente: SANCHEZ

3. Desarrollo Ingenieril

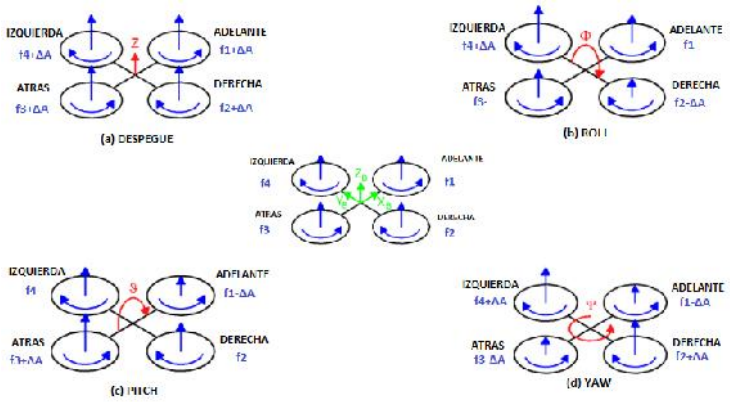
En este apartado se explicará paso a paso el desarrollo empleado para la elaboración del proyecto, hablando así de temas como el estudio del modelo matemático del quadrotor, los métodos empleados para los controladores, el procesamiento de imagen para detectar el objetivo y el tipo de comunicación utilizada por el dispositivo. En este capítulo se presenta los conceptos teóricos del proyecto tales como el modelo matemático del quadrotor, visión por computador en serie y en paralelo, controladores propuestos y la comunicación del quadrotor con la estación remota.

3.1. Descripción de Funcionamiento y Modelamiento del Quadrotor

3.1.1. Descripción de Funcionamiento

Un quadrotor (quadcopter) es un helicóptero que tiene cuatro rotores igualmente espaciados, por lo general localizados en la esquinas de la estructura. El movimiento del quadrotor se origina a partir de los cambios de velocidades de los rotores. Cada rotor consta de un motor eléctrico de corriente continua, un mecanismo de engranaje y un rotor de palas. Para lograr movimiento hacia adelante la velocidad del rotor trasero debe aumentar y, al mismo tiempo, la velocidad del rotor delantero debe ser disminuida. El desplazamiento lateral se ejecuta con el mismo procedimiento, pero usando los rotores de la derecha y la izquierda. El movimiento de guiñada (yaw) se obtiene a partir de la diferencia en el par de torsión entre cada par de rotores, se acelera los dos rotores con sentido horario mientras se desacelera los rotores con sentido anti-horario, y vice-versa. Esto se explicara con mas claridad en el siguiente esquema:

Figura 16: Movimientos básicos del quadrotor



Fuente: Ar Drone Developer Guide

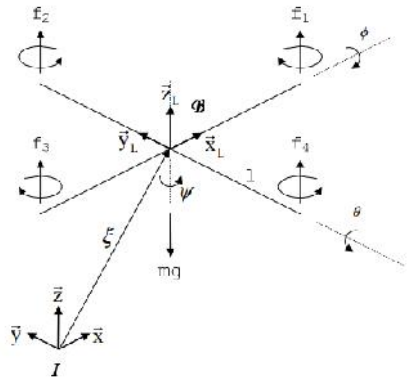
3.1.2. Modelado Matemático Del Sistema

Tabla 1: Nomenclatura utilizada en el modelo

Variable	**Símbolo**
Fuerzas aerodinámicas producidas	Ω_i
Velocidad angular del rotor	ω
Momento de inercia rotacional	J_R
Distancia entre cetro de la estructura a rotores	l
Momento de inercia en el cuerpo rigído	J
Ángulo Pitch	θ
Ángulo Roll	ϕ
Ángulo Yaw	ψ
Velocidad angular Pitch	q
Velocidad angular Roll	p
Velocidad angular Yaw	r
Matriz de coseno directa	R_I
Vector de velocidad en cada eje incercial	v
Vector de velocidad traslacional	V
Masa del quadrotor	m
Matriz identidad	$I_{3\times3}$
Vector de posicion respecto a la inercia	ξ
Vector de ángulos Euler	η
Fuerzas externas al cuerpo	F
Pares externos al cuepor	τ
Vector de fuerzas aerodinámicas	A_T
Vector de pares aerodinámicos	A_R
Gravedad	g
Coeficiente de empuje de rotores	b

En esta sección se desarrollará el modelado basado en leyes físicas que describan la posición y orientación del quadrotor. El modelado cinemático y dinámico del quadrotor se presenta bajo leyes de Newton. Para determinar el modelo dinámico se supone como un cuerpo rígido, sujeto a una fuerza de empuje y tres momentos pares. En la figura 16 se podrá ver las fuerzas que se ejercen en cada una de la hélices lo cual le permitirá realizar distintos movimientos.

Figura 17: Sistema de coordenadas del quadrotor



Fuente: Raffo

Para el estudio del comportamiento del quadrotor [17] supone una estructura y hélices rígidas, una estructura simétrica, el centro de masa y el origen de coordenadas se asume coincidente y por último el empuje y la resistencia al avance son proporcionales al cuadrado de la velocidad de las hélices.

Con estas consideraciones se establece la dinámica del sistema como un cuerpo rígido, añadiéndole las fuerzas aerodinámicas producidas Ω_i por los cambios de velocidades de los rotores; en este modelo se debe considerar efectos giroscópicos. Estos efectos son descritos por [18] los cuales son mostrados en el cuadro 1, donde C representa términos constantes, ω es la velocidad del rotor, J_R es el momento de inercia rotacional del motor alrededor de su eje, *l* es la distancia entre el centro de la estructura a los rotores, *J* es el momento de inercia en el cuerpo rígido y $\phi, \quad \theta \quad y \quad \psi$ son los ángulos Euler.

Tabla 2: Efectos físicos actuantes en el Quadrotor

EFECTOS	F**UENTES**	**FORMULACIÓN**
Efectos Aerodinámicos	Rotación de motores Giro de helicés	$C\Omega^2$
Pares Inerciales Opuestos	Cambio en la velocidad de rotación de los motores	$J_R\dot{\Omega}$
Efectos de la gravedad	Posición del centro de masa	l
Efectos Giroscópicos	Cambio en la orientación del cuerpo rígido	$J\theta\psi$
	Cambio en la orientación del plano de los rotores	$J_R\Omega\theta, \phi$
Fricción	Todos lo movimiento del quadrotor	$C\dot{\phi}, \dot{\theta}, \dot{\psi}$

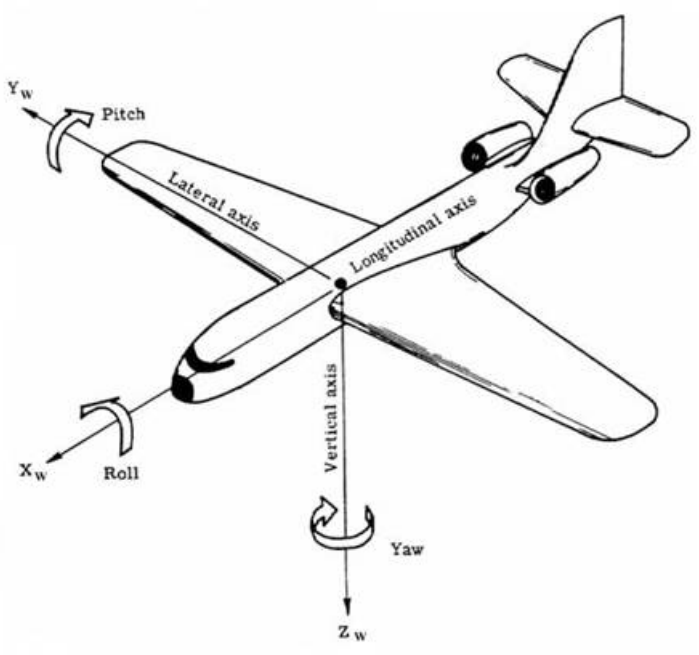
3.1.3. Modelado Cinemático

El movimiento en la estructura consta de seis grados de libertad: tres de ellos definen la posición de un punto de referencia en el cuerpo, y los otros tres definen la orientación del cuerpo. La orientación de un cuerpo rígido puede ser parametrizado usando muchos métodos como Cuaternios y los ángulos de Euler. Los águlos Euler son muy utilizados en ingeniería espacial, esos describen la orientación en el espacio respecto a su referencia fija.

Por lo tanto, los ángulos Euler son tres ángulos usados para describir una rotación general en el espacio Euclídeo [1] tridimensional a través de tres rotaciones sucesivas en torno de ejes del sistema móvil en el cual están definidos. Estos ángulos son alabeo, cabeceo y giñada, aunque son mas conocidos por sus nombres en ingles (roll, pitch y yaw respectivamente).La matriz de rotación se obtiene por tres rotaciones actuantes en el cuerpo rígido, la primera rotación de *x* esta dada por el angulo roll $(-\pi < \phi < \pi)$, la segunda esta dada en el *y* por el ángulo pitch $(-\pi/2 < \theta < \pi/2)$ y por ultimo la rotación en *z* por el ángulo yaw $(-\pi < \psi < \pi)$.

[1] Un espacio euclídeo es un espacio vectorial completo dotado de un producto interno (lo cual lo convierte además en un espacio normado, un espacio métrico y una variedad riemanniana al mismo tiempo)

Figura 18: Movimientos básicos del quadrotor



Fuente: Ar Drone Developer Guide

Así, la configuración de la rotación del cuerpo rígido en el espacio es realizada a través de tres rotaciones sucesivas:

$$R(x,\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{1}$$

$$R\left(y,\theta\right) = \begin{pmatrix} cos\left(\theta\right) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin\left(\theta\right) & 0 & cos\left(\theta\right) \end{pmatrix} \tag{2}$$

$$R(z,\psi) = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

La matriz de rotación completa con respecto a la inercia **I**, es llamada Matriz de Coseno Directa, descrita por [19]:

$$R_I = R\left(x,\phi\right) R\left(y,\theta\right) R\left(z,\psi\right) \tag{4}$$

$$R_I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

$$R_I = \begin{pmatrix} cos\psi cos\theta & cos\psi sin\theta sin\phi - sin\psi cos\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{pmatrix} \quad (6)$$

Gracias a la ortogonalidad la matriz de rotación respecto al eje de coordenadas del cuerpo ***B***, es la transpuesta de R_I y esta expresada de la siguiente forma:

$$R_B = \begin{pmatrix} cos\psi cos\theta & sin\psi cos\theta & -sin\theta \\ cos\psi sin\theta sin\phi - sin\psi cos\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\theta sin\phi \\ cos\psi sin\theta cos\phi + sin\psi sin\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\theta cos\phi \end{pmatrix} \quad (7)$$

La ecuaciones cinemáticas de rotación del vehículo que establecen las relaciones entre velocidades angulares, se obtienen mediante el siguiente análisis:

$$\dot{R}_I = R_I S(\omega) \quad (8)$$

donde $\omega = \begin{pmatrix} p & q & r \end{pmatrix}^T$ son las velocidades angulares en el sistema de coordenadas del cuerpo rígido y $S(\omega) \quad (S(\omega)(\cdot) = \omega \times \cdot)$ es la siguiente matriz anti-simétrica [20]:

$$S(\omega) = \begin{pmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{pmatrix} \quad (9)$$

Al operar matemáticamente la ecuación (8) con las ecuaciones (6) y (9) se obtiene la siguiente relación:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sen(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sen(\phi) \\ 0 & -sen(\phi)arcos(\theta) & cos(\phi)arcos(\theta) \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (10)$$

La relación entre las velocidades angulares en el sistema fijo y la variación en el tiempo de los ángulos Euler se obtiene a través de la inversión del Jacobiano representado por la ecuación (10), y se viene dada por:

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} 1 & 0 & -sen(\theta) \\ 0 & cos(\phi) & sen(\phi)cos(\theta) \\ 0 & -sen(\phi) & cos(\phi)cos(\theta) \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \tag{11}$$

La cinemática translacional del cuerpo esta dada por los componentes de la velocidades angulares en los tres ejes: velocidad angular en roll (p), velocidad angular en pitch (q), y velocidad angular en yaw (r). Estas velocidades rotacionales son producidas gracias a los pares ejercidos sobre el sistema ligado al cuerpo del quadrotor causadas por fuerzas externas, las cuales definen los momentos en cada eje: momento de balanceo (L), momento de cabeceo (M),y momento de guiñada (N) sobre casa eje respectivamente.

La cinemática traslacional esta dada por componentes de la velocidad $v = (u_0 \quad v_0 \quad w_0)^T$ en los tres ejes inerciales con relación a la velocidad absoluta del helicoptero la cual se expresa en **B**, $V = (u_L \quad v_L \quad w_L)^T$. Las velocidades **v** y **V** estan relacionadas por la siguiente expresión:

$$v = R_I V \tag{12}$$

3.1.4. Modelamiento Dinámico

La dinámica del modelo esta descrita por movimientos translacionales y rotacionales, esta se plantea mediante la formulación Newton-Euler.

Formulación Newton-Euler
En esta sección se obtendrán las ecuaciones dinámicas del quadrotor mediante formulación de Newton-Euler, la justificación completa de estas están descritas por [21].

Las ecuaciones dinámicas están sujetas a las fuerzas externas aplicadas en el sistema de coordenadas del cuerpo, se pueden obtener a través de la formulación de Newton-Euler como sigue:

$$\begin{bmatrix} mI_{3\times3} & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega & \times & mV \\ \omega & \times & J\omega \end{bmatrix} = \begin{bmatrix} F & + & F_d \\ \tau & + & \tau_d \end{bmatrix} \tag{13}$$

Donde J es la matriz de inercia, $I_{3\times3}$ es la matriz identidad, V es el vector de velocidad translacional en el cuerpo, ω es el vector de velocidad angular en el cuerpo y m es la masa del vehículo.

Descrito lo anterior se puede suponer la matiz de inercia como diagonal:

$$J = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \tag{14}$$

Se da a consideración un vector de estados $\begin{bmatrix} \xi & v & \eta & \omega \end{bmatrix}^T$ donde ξ y v representan la posición y velocidad con respecto a la inercia, $\eta = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}$ y ω son las velocidades angulares expresadas en B, se puede escribir las ecuaciones de movimiento del cuerpo rígido de la siguiente forma:

$$\begin{gathered} \dot{\xi} = v \\ m\dot{v} = R_I F_b \\ \dot{R}_I = R_I S(\omega) \\ J\dot{\omega} = -\omega \times J\omega + \tau_b \end{gathered} \tag{15}$$

Como se menciono con anterioridad, el quadrotor es un sistema subactuado con seis grados de libertad y cuatro actuadores (las fuerzas y momentos producidos por los cuatro rotores).

Por otra parte F_b y τ_b actúan en B y estos son las fuerzas y pares externos aplicados al cuerpo, y consisten en su propio peso, en el vector de fuerzas aerodinámicas, en el empuje y en los pares desarrollados por los cuatro rotores. Estas fuerzas y pares pueden expresarse de esta forma:

$$\begin{gathered} R_I F_b = -mg \cdot E_3 + R_{IE_3} \left(\sum_{i=1}^{4} b\Omega_i^2 \right) + A_T \\ \tau_b = -\sum_{i=1}^{4} J_R(\omega \times E_3) \cdot \Omega_i + \tau_a + A_R \end{gathered} \tag{16}$$

Con las ecuaciones de fuerzas y pares (16), el modelo dinámico (15) se reescribe de la siguiente forma:

$$\begin{gathered} \dot{\xi} = v \\ \dot{v} = -g \cdot E_3 + R_{IE_3} \frac{b}{m} \left(\sum_{i=1}^{4} \Omega_i^2 \right) + \frac{A_T}{m} \\ \dot{R}_I = R_I S(\omega) \\ J\dot{\omega} = -\omega \times J\omega - \sum_{i=1}^{4} J_R(\omega \times E_3) \cdot \Omega_i + \tau_a + A_R \end{gathered} \tag{17}$$

donde:

- Los vectores $A_T = \begin{bmatrix} A_x & A_y & A_z \end{bmatrix}^T$ y $A_R = \begin{bmatrix} A_p & A_q & A_r \end{bmatrix}^T$ son las fuerzas y pares aerodinámicos actuantes en el vehículo, son calculadas a partir de los coeficientes aerodinámicos C_i como $A_i = \frac{1}{2}\rho_{aire} C_i W^2$ donde ρ_{aire} es la densidad del aire y W es la velocidad del vehículo respecto al aire[22].

- g es la gravedad ($g = 9{,}81m/seg^2$).
- J_R representa el momento de inercia rotacional de los rotores alrededor de su eje.
- b es el coeficiente de empuje aplicado por los rotores.
- Ω_i es la velocidad angular de cada motor.

Como lo describe [23] la fuerza o entrada de control principal es U_1, esta es la sumatoria de las fuerzas traslacionales actuantes, esta se evidencia en la ecuación (16) la cual se esta compuesta por el empuje total generado por la suma de los cuatro rotores, por la fuerza gravitacional y por la fuerza aerodinámica.

$$U_1 = \left(\sum_{i=1}^{4} fi\right) = \left(\sum_{i=1}^{4} b\Omega_i^2\right) \tag{18}$$

El par τ_a de la ecuación (16) es el vector de pares de control aplicados al quadrotor. Este se obtiene a través del esfuerzo de torsión τ_M, generado por cada motor eléctrico considerando la dinámica de cada disco del motor como un sistema desacoplado en la variable generalizada Ω_i, que detona la velocidad angular de un motor alrededor de su eje. El esfuerzo de tensión del motor es opuesto a la fricción aerodinámica del motor $\tau_{drag} = k_\tau \Omega_i^2$, donde $k_\tau > 0$ es una constante. Así, a través de la segunda Ley de Newton se obtiene [23]:

$$J_R \dot{\Omega}_i = -\tau_{drag} + \tau_{M_i} \tag{19}$$

Cuando $\dot{\Omega}_i = 0$ se tiene que:

$$\tau_{M_i} = \tau_{drag} = k_\tau \Omega_i^2 \tag{20}$$

El momento aplicado en el cuerpo rígido a lo largo de un eje es la diferencia entre el momento generado por cada motor en el otro eje. Los movimientos de cabeceo (pitch), balanceo (roll) y guiñada (yaw) son generado gracias a las diferencias entre las relaciones de empujes como ya se había comentado anteriormente. Estos movimientos deben ser logrados con la fuerza principal constante [23]. Así, el par de control aplicado en los tres ejes viene dado por:

$$\tau_a = \begin{bmatrix} (f_2 - f_4)l \\ (f_3 - f_1)l \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} = \begin{bmatrix} lb(\Omega_2^2 - \Omega_4^2) \\ lb(\Omega_3^2 - \Omega_1^2) \\ k_\tau(\Omega_1^2 + \Omega_3^2 - \Omega_2 - \Omega_4^2) \end{bmatrix} \begin{bmatrix} l \cdot U_2 \\ l \cdot U_3 \\ U_{4|} \end{bmatrix} \tag{21}$$

donde l es la distancia entres los rotores y el centro de gravedad.

Cada rotor se puede considerar como un disco rígido rotando alrededor de su eje z con una velocidad Ω_i. El eje de rotación del motor se mueve con la velocidad angular del eje de referencia, lo cual produce movimientos giroscópicos que se describen matematicamente como se presente acontinuación:

$$\tau G_a = -\sum_{i=1}^{4} J_R(\omega \times E_3) \cdot \Omega_i \tag{22}$$

Determinando las ecuaciones (11), (12), (18) y (21) se define un nuevo vector de estados como [22]:

$$\zeta = [x \quad y \quad z \quad u_0 \quad v_0 \quad w_0 \quad \phi \quad \theta \quad \psi \quad p \quad q \quad r]^T \tag{23}$$

La ecuación de movimiento (17) se puede reescribir de la siguiente forma:

$$\dot{\zeta} = \begin{cases} \dot{x} = u_0 \\ \dot{y} = v_0 \\ \dot{z} = w_0 \\ \dot{u_0} = \frac{1}{m}(cos\psi sin\theta cos\phi + sin\psi sin\phi) \cdot U_1 + \frac{A_x}{m} \\ \dot{v_0} = \frac{1}{m}(sin\psi sin\theta cos\phi - cos\psi sin\phi) \cdot U_1 + \frac{A_y}{m} \\ \dot{w_0} = -g + \frac{1}{m}(cos\theta cos\phi) \cdot U_1 + \frac{A_z}{m} \\ \dot{\phi} = p + qsin\phi tan\theta + rcos\phi tan\theta \\ \dot{\theta} = qcos\phi - rsin\phi \\ \dot{\psi} = qsin\phi sec\theta + rcos\phi sec\theta \\ \dot{p} = \frac{(I_{yy}-I_{zz})}{I_{xx}} qr - \frac{J_R\Omega}{I_{xx}} q + \frac{l}{I_{xx}} U_2 + \frac{A_p}{I_{xx}} \\ \dot{q} = \frac{(I_{zz}-I_{xx})}{I_{yy}} pr - \frac{J_R\Omega}{I_{yy}} p + \frac{l}{I_{yy}} U_3 + \frac{A_q}{I_{yy}} \\ \dot{r} = \frac{(I_{xx}-I_{yy})}{I_{zz}} pq + \frac{l}{I_{zz}} U_4 + \frac{A_r}{I_{zz}} \end{cases} \tag{24}$$

La ecuación diferencial no lineal (24) se puede escribir en una forma mas compacta:

$$\dot{\zeta} = f(\zeta) + \sum_{i=1}^{4} g_i(\zeta) U_i \tag{25}$$

donde:

$$f(\zeta) = \begin{bmatrix} u_0 \\ v_0 \\ w_0 \\ A_x/m \\ A_y/m \\ A_z/m \\ -g + \frac{A_z}{m} \\ p + qsin\phi tan\theta + rcos\phi tan\theta \\ qcos\phi - rsin\phi \\ qsin\phi sec\theta + rcos\phi sec\theta \\ \frac{(I_{yy}-I_{zz})}{I_{xx}}qr - \frac{J_R\Omega}{I_{xx}}q + \frac{A_p}{I_{xx}} \\ \frac{(I_{zz}-I_{xx})}{I_{yy}}pr - \frac{J_R\Omega}{I_{yy}}q + \frac{A_q}{I_{yy}} \\ \frac{(I_{xx}-I_{yy})}{I_{zzx}}pq + \frac{A_r}{I_{zz}} \end{bmatrix}$$

$$g_1(\zeta) = [0 \quad 0 \quad 0 \quad g_1^4 \quad g_1^5 \quad g_1^6 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

$$g_2(\zeta) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{xx}} & 0 & 0 \end{bmatrix}^T$$

$$g_3(\zeta) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{yy}} & 0 \end{bmatrix}^T$$

$$g_4(\zeta) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix}^T$$

Ya que:

$$g_1^4 = \frac{1}{m}(cos\psi sin\theta cos\phi + sin\psi sin\phi)$$

$$g_1^5 = \frac{1}{m}(sin\psi sin\theta cos\phi - cos\psi sin\phi)$$

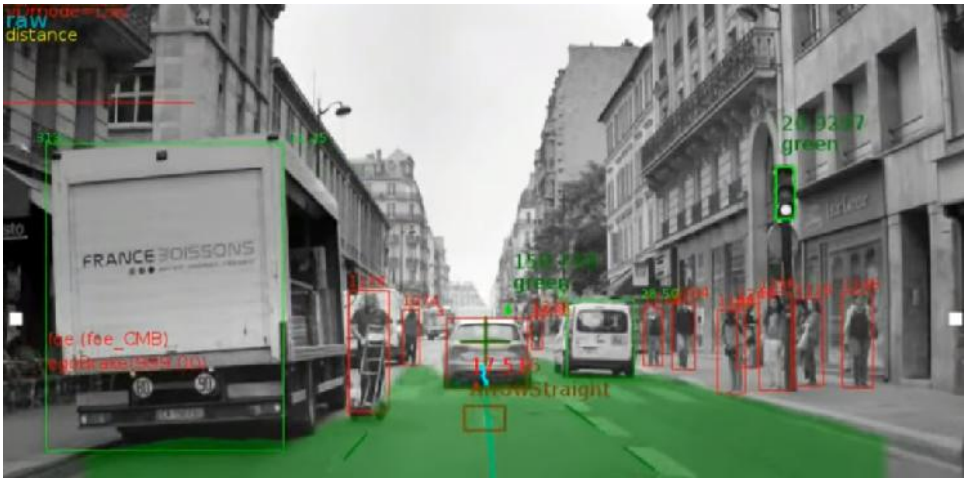
$$g_1^6 = \frac{1}{m}(cos\theta cos\phi)$$

3.2. Visión por Computador

En este capítulo se presentan las técnicas de procesamiento de imágenes usadas en la visión de computador. La visión por computador tiene como iniciativa automatizar e integrar una amplia gama de procesos y representaciones utilizados para la percepción (Inspeccionar el ambiente que le rodea y hacerse una idea propia de él) [24]. Forsyth [25], describe visión por computador como la extracción de las características del mundo a partir de imágenes o secuencias de imágenes y Shapiro[26], la define como la toma de decisiones sobre objetos físicos reales y escenas basado en imágenes.

Un ejemplo de toma de decisiones es la incorporación usada por el vehículo autónomo Tesla, la cual detecta el carril de conducción, peatones, otros vehículos, entre otros parámetros para su navegación autónoma como se muestra en la figura 19. Los datos de entrada del vehículo son sus cámaras, sensor de radar, sensor ultrasonido y un GPS[27].

Figura 19: Tesla visión por computador



Fuente: Tesla responds to Mobileye's comments on Autopilot, confirms new in-house 'Tesla Vision' product, 2016. URL: *https://electrek.co/2016/09/15/tesla-vision-mobileye-tesla-autopilot/*

Procesamiento de Imágenes

Es el estudio por medio de la implementación de un algoritmo que toma una imagen de entrada y devuelve una imagen de salida modificada. Este puede incluir, detección de bordes, eliminación de ruido, detección de objetos, entre otras técnicas. En este proyecto se desarrollan los conceptos de procesamiento de imágenes por CPU y procesamiento de imágenes por GPU. Para procesamiento de imágenes por CPU se hará uso de las librerías OpenCV y para procesamiento con GPU o paralelo se utiliza arquitectura CUDA.

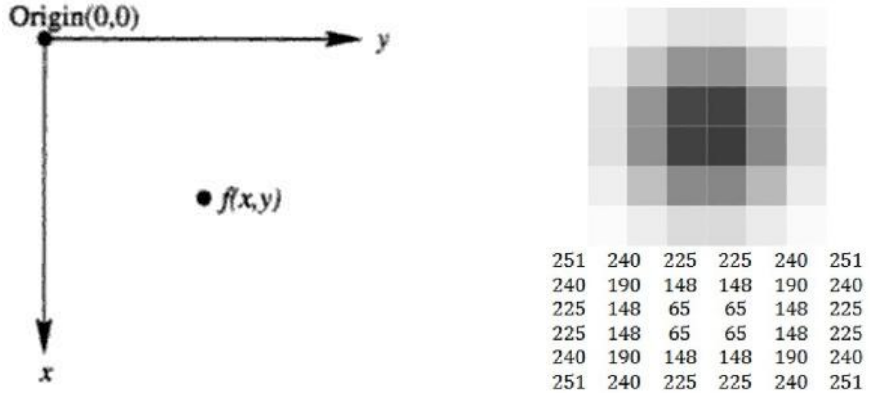
Procesamiento de Imágenes con CPU

3.2.1. Imagen Digital

Una imagen digital contiene un número fijo de columnas y filas de pixeles. Un pixel contiene un valor normalmente de 0 a 255 que representa el brillo en una imagen. Así, por ejemplo, en una imagen en escala de grises el valor cero puede

ser el valor más oscuro y el 255 representa el valor con más brillo o vice versa como se muestra en la figura 20b. En una imagen de color RGB un pixel contiene tres valores, uno para rojo, uno para verde y uno para azul en el espacio de color RGB. En la figura 20a se muestra la representacion de una imagen digital en sus sistema de coordenadas.

Figura 20: (a)Sistema de coordenadas para la representación de una imagen digital, (b) Imagen de 6x6 en escala de grises donde 0 es negro y 255 es blanco.



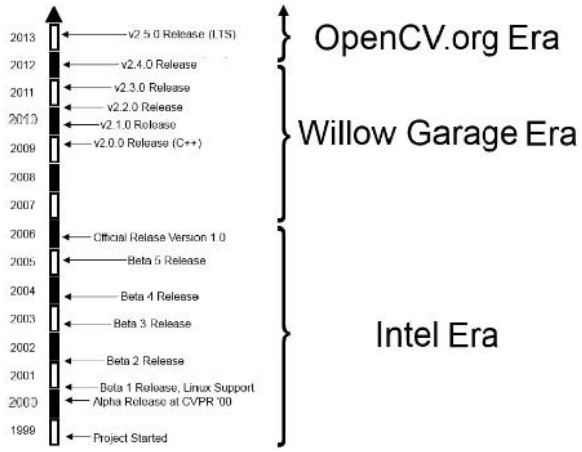
(a) (b)

Fuente: (a)Digital Image Processing Algorithms and Applications Capitulo 1 Pg. 9 (b)Exploring Image Processing and Image Restoration Techniques

3.2.2. OpenCV

OpenCV (Open Source Computer Vision) es una librería libre de visión por computador desarrollada en 1999 por INTEL bajo la licencia BSD. El objetivo principal era proveer una librería para procesamiento de imágenes en tiempo real y desde entonces ha sido una herramienta fundamental para la visión por computador. Así mismo, por el crecimiento en el campo de la robótica y la llegada de procesadores multinucleo ha impulsado mucho el uso y desarrollo de la librería. Hubo un tiempo en el cual INTEL no estaba desarrollando la librería lo cual Willows Garage continuo su desarrollo junto a Itseez. En el año 2012, la fundación OpenCV.org asumió la tarea de mantener un sitio web de soporte para desarrolladores y usuarios. En la figura 21 se presenta la evolución de OpenCV.

Figura 21: Evolución OpenCV

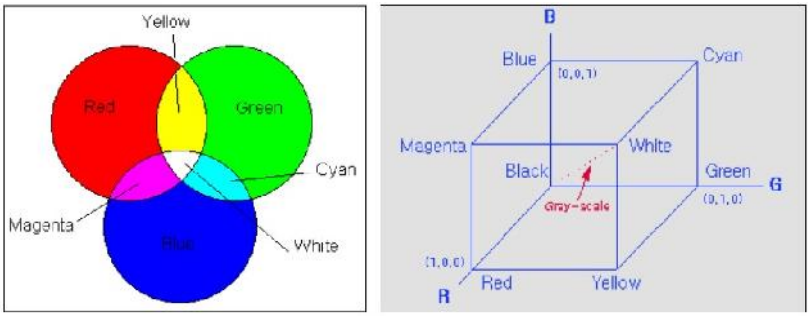


Fuente: Learning OpenCV Computer Vision in C++ with the OpenCV Library Capitulo 1 Pg. 6

3.2.3. Espacio de Color

El espacio de color más común es RGB (Red-Green-Blue). Este es un espacio de color aditivo, es decir se le añade ondas de luz del rojo, verde y azul para tener como resultado un color final como se muestra en la figura 22. Cuando estos tres colores faltan el color resultante es negro y cuando estos colores están en su máxima intensidad el color es blanco [28]. Este modelo de color es el más predominante en gráficos de computadora porque las pantallas usan rojo, verde y azul para crear el color deseado. Dado a que este espacio de color está representado por la cantidad de luz reflejado en un objeto, a menudo es difícil identificar el objeto en condiciones diferentes de iluminación.

Figura 22: Modelo RGB

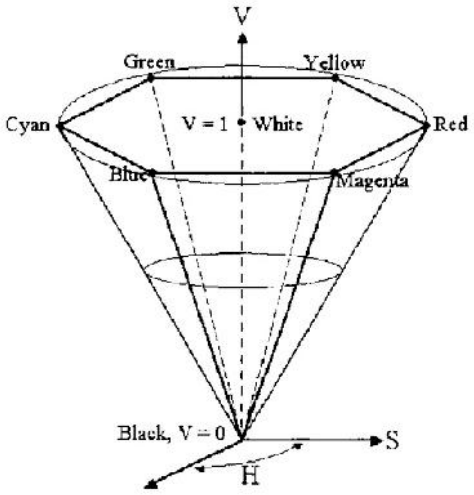


Fuente: Digital Image Representation and Color Fundamentals. URL: *http://me.umn.educoursesme5286visionNotes/2015/ME5286-Lecture3.pdf*

Por otra parte, existe otro espacio de color llamado HSV el cual describe los

colores como los percibe el ser humano. HSV por sus siglas significa hue (H), saturación (S) y intensidad (V). El componente hue es un atributo de la percepción humana y puede ser describía como rojo, verde, azul, morado y amarillo como hue primarios. El componente de saturación está dado por el colorido de los colores y el componente de intensidad proporciona una medida del brillo de los colores. La representación del espacio de color está dada por una figura cónica como se ve en la figura 23, donde el componente (V) varia en el eje vertical, el componente (H) varia sobre la periferia del circulo y el componente (S) varia a lo largo de la distancia radial[29]. Este espacio de color es mucho más robusto al cambio de iluminación lo cual lo hace ideal para procesamiento de imágenes.

Figura 23: Representación espacio de color HSV



Fuente: Image Processing: Principles and Applications Capitulo 3 Pg. 44

Dicho lo anterior, como la mayoría de imágenes están en el espacio de color RGB existen varias transformaciones para convertir la imagen del espacio RGB a el espacio HSV. La transformación más simple está dada por las siguientes ecuaciones[30]:

$$H = tan\left[\frac{3(G-B)}{(R-G)+(R-B)}\right], S = 1 - \frac{min(R,G,B)}{V}, V = \frac{R+G+B}{3} \tag{26}$$

Otro método más popular para hacer la transformación de RGB a HSV es normalizando los pixeles r, g b de la forma [30]

$$r = \frac{R}{R+G+B}, g = \frac{G}{R+G+B}, b = \frac{B}{R+G+B} \tag{27}$$

Después, los valores H, S y V son calculados con:

$$V = max(r, g, b) \tag{28}$$

$$S = \begin{cases} 0 & if \;\; V = 0 \\ V - \frac{min(r,g,b)}{V} & if \;\; V > 0 \end{cases} \tag{29}$$

$$H = \begin{cases} 0 & if \;\; S = 0 \\ \frac{60*(g-b)}{S*V} & if \;\; V = r \\ 60 * [2 + \frac{(b-r)}{S*V}] & if \;\; V = g \\ 60 * [4 + \frac{(r-g)}{S*V}] & if \;\; V = b \end{cases} \tag{30}$$

$$H = H + 360 \quad if \;\; H < 0 \tag{31}$$

Con la librería OpenCV la función *cv::cvtColor(src, dst, RGB2HSV)* convierte automáticamente una imagen de RGB a HSV, donde src es la imagen de entrada, dst la imagen de destino y RGB2HSV el código indicando la transformación a HSV. Como el valor de hue es de 0 a 360, este puede causar un error cuando la imagen tiene valores de 8 bits. Para evitar este inconveniente y manipular el componente hue, este se divide en 2. La diferencia entre estos espacios de color se evidencia en la figuras 24a y 24b.

Figura 24: Diferencia entre espacio de color RGB y HSV

(a) RGB (b) HSV

Fuente: Propia del autor.

3.2.4. Segmentación

Segmentación es la tarea de encontrar grupos de pixeles que van juntos, es decir la identificación de regiones en una imagen. Como lo menciona Bernardes[9], el proceso de segmentación se destina para dividir una imagen digital en partes constituyentes que sirven para facilitar el análisis de la imagen. En procesamiento de imágenes, esta es uno de los problemas más antiguos y más estudiado[31]. Puesto que hay diversos tipos de segmentación, una de ellas utilizando imágenes en escala de grises. Esta segmentación se basa es convertir una imagen de escala de grises a una imagen en blanco y negro donde se escoge un valor de umbralización (El valor en cual el histograma de una imagen se divide en dos). Con este valor de umbralización, todos los niveles de grises por debajo del valor establecido se clasificarán como negro (0), y los valores por encima serán blancos (1). Un ejemplo de esta umbralizaciòn se evidencia en la figura 25 donde la 25a es la imagen original y la 25b es la imagen resultante con un valor establecido de 110.

Figura 25: Umbralizacion en escala de grises

(a) (b)

Fuente: Propia del autor

Por otra parte, la técnica para lograr una segmentación de color eficaz es transformando la imagen de RGB a HSV. Ya transformada la imagen se varia el componente HUE el cual es el que caracteriza el color. Ya cuando se obtiene el color deseado se varia la saturación (S) y la intensidad (V) dependiendo la iluminación en el ambiente. En la figura 8 se obtuvo la segmentación del color amarillo donde la figura 26a es la imagen original y la figura 26b es la imagen resultante donde el objeto es blanco y la región de no interés es lo negro.

Figura 26: Segmentación por color

(a) (b)

Fuente: Propia del autor

3.2.5. Operaciones Morfológicas

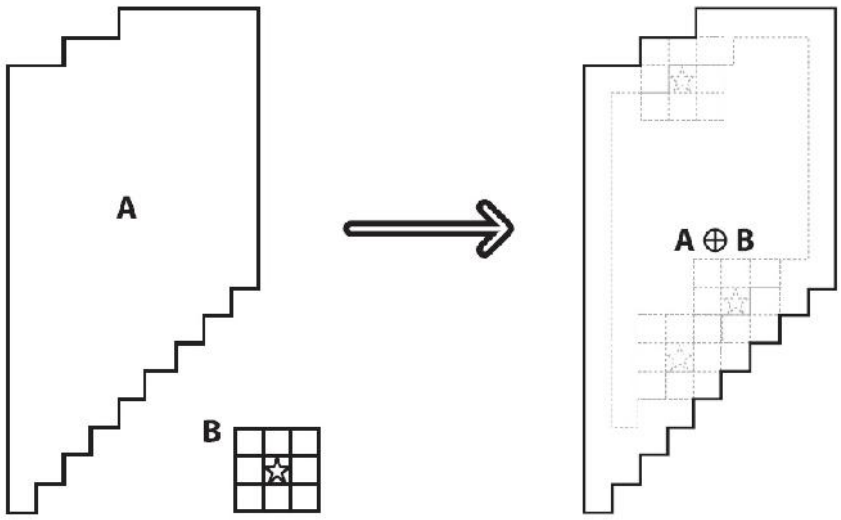
Las operaciones morfológicas tienen la característica de aplicación como eliminar ruido, asilar elementos individuales y unir elementos dispares en una imagen. Shapiro [26], refieren la palabra morfología como una forma y estructura de un objeto; en visión por computador puede utilizarse para referirse a una región. La mayoría de las operaciones morfológicas son implementadas en imágenes binarias después de una umbralizaciòn, pero pueden ser también implementadas en imágenes de intensidad[32].Las operaciones de morfología binaria tienen como entrada un elemento estructurante, lo que usualmente es una imagen binaria más pequeña. Este elemento estructurante puede ser de cualquier tipo de figura o tamaño. El propósito del elemente estructurante es examinar la imagen binaria y contiene un pixel de origen el cual se computan las translaciones para ampliar una región por la forma de la figura o verificar si la figura cabe en una región.

Una de las transformaciones morfológicas más comunes es la dilatación. El efecto de la dilatación es engrandar la región clara de la imagen ya que al examinar la imagen con el elemento estructurante si halla un 1 deja el valor máximo en el origen. La representación de la dilatación es:

$$A \oplus B$$

Donde A es la imagen binaria y B el elemento estructurante, la figura 27 muestra el efecto de la dilatación.

Figura 27: Representación operación morfológica dilatación



Fuente: Learning OpenCV Capitulo 5 Pg. 116

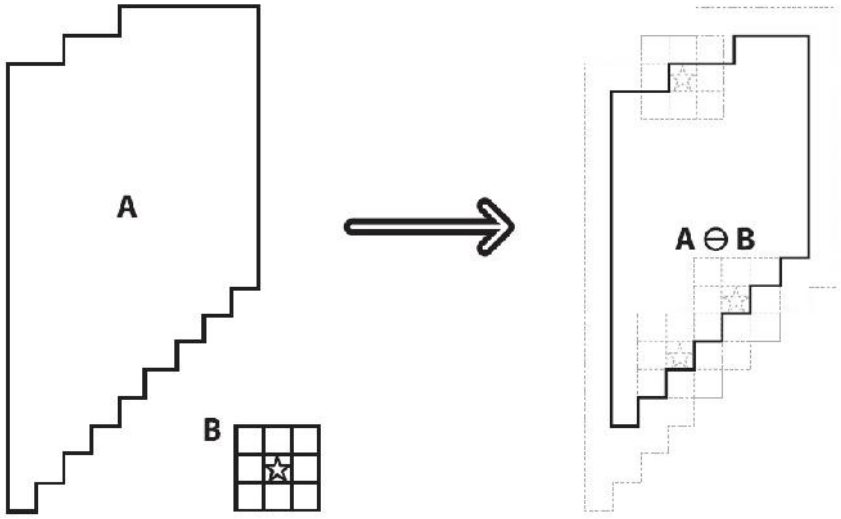
Otra operación morfológica común es la erosión. A comparación de la dilatación,

la erosión reduce la región clara de la imagen. La erosión es una herramienta esencial para remover ruido. La representación de la erosión es:

$$A \ominus B$$

Donde A es la imagen binaria y B el elemento estructurante, la figura 28 muestra el efecto de la erosión.

Figura 28: Representación operación morfológica erosión



Fuente: Learning OpenCV Capitulo 5 Pg.117

Aunque estas operaciones son inversas al aplicar una dilatación o erosión no se puede devolver a la imagen original.

3.2.6. Contornos

Contornos son un conjunto de puntos conectados entre sí, situados en los bordes de los objetos[32]. Los contornos son una herramienta útil para análisis de figuras y detección o reconocimiento de objetos. Para extraer los contornos y tener una buena precisión se utiliza una imagen binaria donde el objeto es de color blanco y el fondo negro. Para la extracción de contornos con OpenCV se utiliza la función *cv::findContours()* y se almacena los contornos en un vector STL.

3.2.7. Momentos

Según Flusser [33], momentos son cantidades escalares utilizadas para caracterizar una función y capturar sus características significativas. Estos han sido usados

en estadística y en estática mecánica para medir la distribución de masa en un cuerpo y está dada por la siguiente ecuación:

$$m_{p,q} = \sum_{i=1}^{n} I(x,y) x^p y^q$$

Donde p es el orden en x y q el orden en y, el orden significa la potencia a la que se toma el componente correspondiente en la suma. La sumatoria es sobre todos los pixeles del límite del contorno denotado como *n* en la ecuación.

Los momentos son una parte fundamental ya permiten calcular el centroide del objeto segmentado lo que nos puede indicar la posición. Para esto se tiene que el significado del momento m_{00} es el área del objeto y m_{10} y m_{01} son los momentos en *x* y *y*. Para el cálculo del centroide es con la siguiente ecuación:

$$x = \frac{m_{10}}{m_{00}}$$

$$y = \frac{m_{01}}{m_{00}}$$

Con el componente *x* y *y* se obtiene las coordenadas del centroide en la imagen.

Procesamiento de Imágenes en Paralelo

3.2.8. Computación en Paralelo

El interés en computación paralela es debido al ralentizado mejoramiento de microprocesadores en los últimos años. Esto es debido al reducir el tamaño de los transistores aumentan los transistores en el circuito integrado lo que los hace más rápido pero su consumo de energía aumenta de igual forma. Este alto consumo de energía es calor disipado y cuando el circuito se calienta se vuelve poco fiable[34]. Más aun, en el campo de procesamiento de imágenes el cual la mayoría de los algoritmos son de alto nivel que requieren una potencia computacional significativa que no puede ser proporcionada por un uniprocesador pero si por una arquitectura paralela[35]. Otro factor importante en el interés por la computación en paralelo es el costo del hardware que ha disminuido constantemente[36].

El paralelismo es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente. En este proyecto se cubrirá computación en paralelo con GPU donde se implementará con arquitectura CUDA. Se propone un algoritmo con cuatro filtros, como el filtro gaussiano, el filtro gradiente sobel, reconstrucción morfológica H-MIN y la transformada watershed.

3.2.9. Diferencia entre CPU y GPU

La CPU y la GPU se diferencia en su arquitectura. La CPU está diseñada para hacer tareas pequeñas pero complejas, mientras que la GPU está diseñada para hacer muchas tareas simples[37]. Otra forma de diferenciar estos dispositivos es en *latency* y *throughput*. *Latency*, es la cantidad de tiempo para completar una tarea y *throughput* es las tareas completadas por unidad de tiempo. La CPU está diseñada para tener una mejor *latency* y la GPU una mejor *throughput*. Así, por ejemplo, en una operación de elevar al cuadrado un conjunto de números del 0 al 63, en una CPU cada operación tarda 2*ns* y el total se demora 128*ns*. En cambio, en una GPU cada operación se tarda 10*ns* pero en total tarda 10*ns* ya que todas las operaciones se hicieron a la misma vez. Otra principal diferencia entre la CPU y la GPU es el gran número de procesadores en cada dispositivo. La figura 29 muestra la diferencia en arquitectura entre estos dispositivos.

Figura 29: Arquitectura CPU y GPU

Fuente: NVIDIA CUDA™ Programming Guide 2.3.1 Capitulo 1 Pg. 3

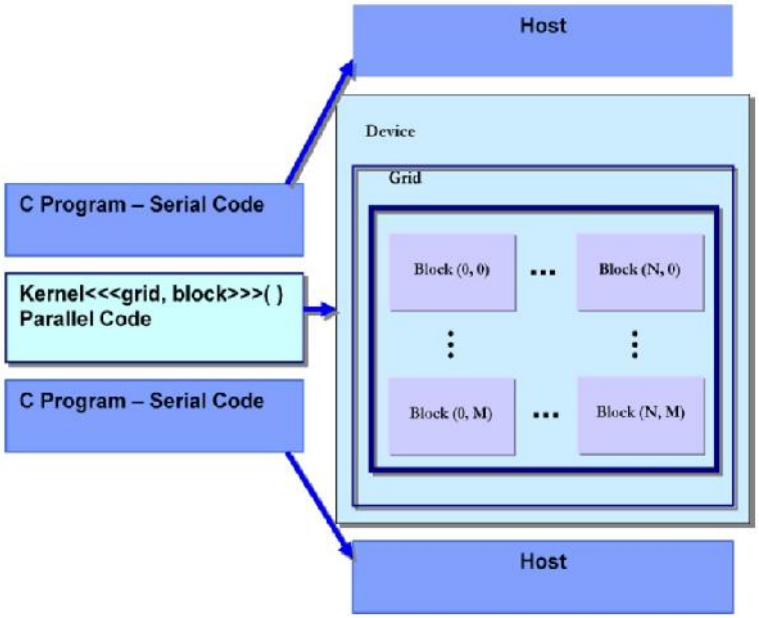
3.2.10. CUDA

CUDA (Compute Unified Device Arquitecture) es una plataforma de computación paralela y modelo de programación que facilita el uso de la GPU para computación de propósito general. El modelo de programación es de SIMD (Single Instruction, Multiple Data) que lo hace perfecto para la aplicación a procesamiento de imágenes ya que se aplica la misma operación en todos los pixeles de la imagen. CUDA contiene unas funciones específicas llamadas *kernels* que son ejecutadas en la GPU un numero N de veces en paralelo utilizando N número de threads. Este modelo de programación permite utilizar el lenguaje de programación C y otros lenguajes como C++, OpenCL y Fortran.

La programación en CUDA es una combinación de ejecuciones en serie y en para-

lelo. Esto se denomina como computación heterogénea ya que hay una interacción entre la CPU y GPU, este se muestra en la figura 30.

Figura 30: Computación Heterogénea



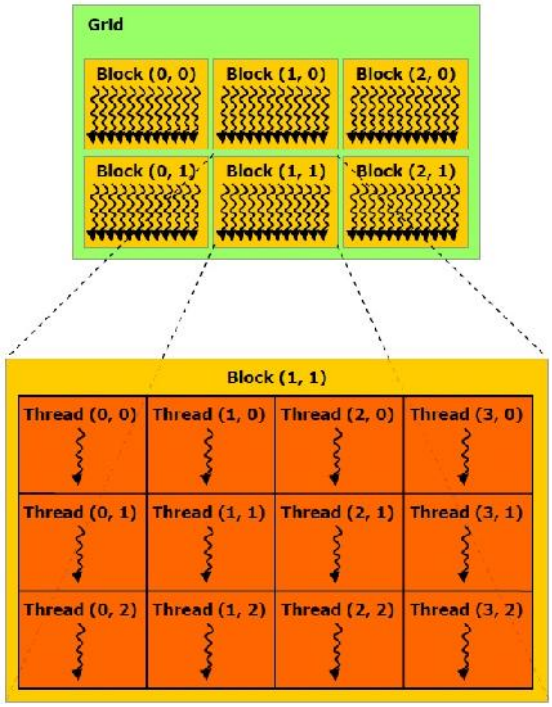
Fuente: An Introduction to GPGPU Programming - CUDA Architecture. URL: *http://www.diva-portal.org/smash/get/diva2:447977/FULLTEXT01.pdf*

Esta interacción entre la CPU y la GPU se da en cuando un código C se ejecuta en serie en la CPU que se denomina como el *host*. La ejecución en paralelo se da cuando el kernel se ejecuta en un conjunto de threads en la GPU; la GPU se denomina como *device*. El kernel solo se puede llamar por la CPU y en la configuración se debe especificar el número de threads en un bloque que se van a ejecutar[38].

Grid, Bloques y Threads

El paralelismo en CUDA es posible con tres elementos estructurados el primero siendo el grid, el segundo bloques y el tercero threads[16]. El grid consiste de bloques de threads de uno, dos o tres dimensiones. Los bloques de threads están divididos también es tres dimensiones y los threads en un bloque están organizados en grupos de 32 threads llamado un *warp*. Todos los threads en un bloque se pueden comunicar entre sí a través de la memoria compartida y se ejecutan en el mismo multiprocesador. Como se mencionó anteriormente, estos elementos son utilizados en la función *kernel* y se puede personalizar para dar una mejor configuración para los requerimientos de la aplicación. Un aspecto importante es tener en cuenta que solo se puede correr un kernel a la vez y no es posible inicializar otro kernel mientras se está ejecutando el anterior.

Figura 31: Estructura de Grid, Bloques y Threads



Fuente: NVIDIA CUDA™ Programming Guide 2.3.1 Capitulo 2 Pg. 10

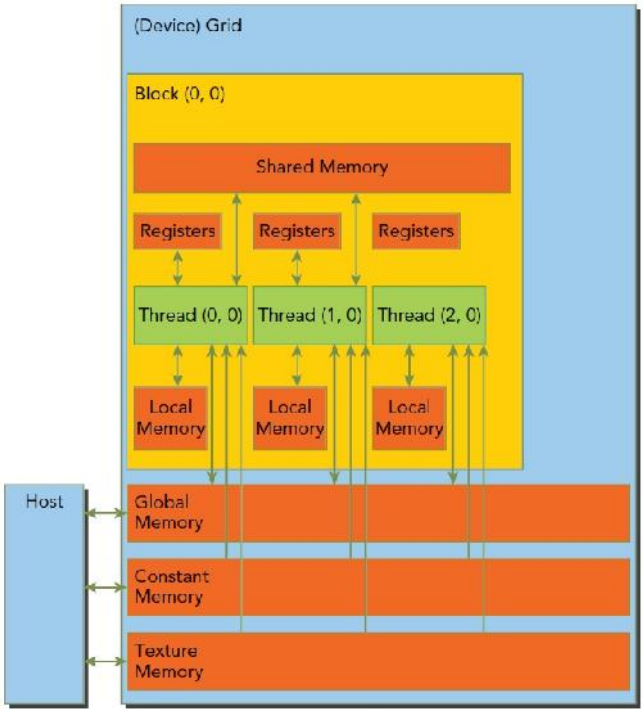
La estructura de los grids, bloques y threads se evidencia en la figura 31. Cada bloque de threads puede contener 512 o 1024 threads, esto depende de la tarjeta gráfica que se está utilizando. Cada thread tiene un identificador que se accede con la variable *threadIdx*, esta variable tiene las tres componentes de las dimensiones. Igual que los threads, los bloques se identifican con la variable *blockIdx* y también tiene componentes en las tres dimensiones. Finalmente, la variable *blockDim* accede al tamaño del bloque[39].

Modelo de Memoria

Tanto las GPUs como las CPUs utilizan principios y modelos similares en el diseño de jerarquías de memoria. La diferencia entre estas dos es que en CUDA se expone más la jerarquía de memoria y le da un control más explícito sobre su comportamiento[40]. El nivel más bajo de jerarquía es la memoria local (Local Memory) que es privada para cada hilo. La memoria local almacena las variables que ya no se pueden almacenar en los registros como matrices muy grandes que ocupa mucha memoria de los registro o matrices las cuales sus índices no se pueden determinar en el momento de compilar. A nivel medio de bloques se tiene la memoria compartida (Shared Memory) que comparte todos los hilos del bloque. Como esta memoria está en un bloque es mucho más rápido acceder a ella que a la memo-

ria global, esta también tiene mejor *throughput* y más bajo *latency* que la memoria global o local. Por último, el nivel más alto es la memoria global (Global Memory) la cual es accesible por los grids y bloque, esta es la memoria más usada en la GPU y con mayor *latency* [40][41]. La figura 32 se presenta el esquema de la memoria.

Figura 32: Esquema de memora



Fuente: Professional CUDA C Programming Capitulo 4 Pg.138

Los otros tipos de memoria son registros, constante y textura los cuales se explican breve mente a continuación.

- **Registros*(Registers)*** - Los registros es el tipo de memoria más rápida en una GPU. Una variable automática declarada en un kernel sin ningún otro calificador de tipo se almacena generalmente en un registro y las variables son privadas para cada thread.
- **Memoria de Texturas*(Texture Memory)*** - La memoria de textura es un tipo de memoria global a la que se accede a través de una caché de sólo lectura. Esta memoria esta optimizada para la localidad espacial 2D, es decir, los threads de un warp que utilice la memoria de texturas para acceder a datos en 2D tendrá un mejor rendimiento.
- **Memoria Constante*(Constant Memory)*** - Igual que la memoria de textura, la memoria constante es un tipo de memoria global solo para lectura, esta

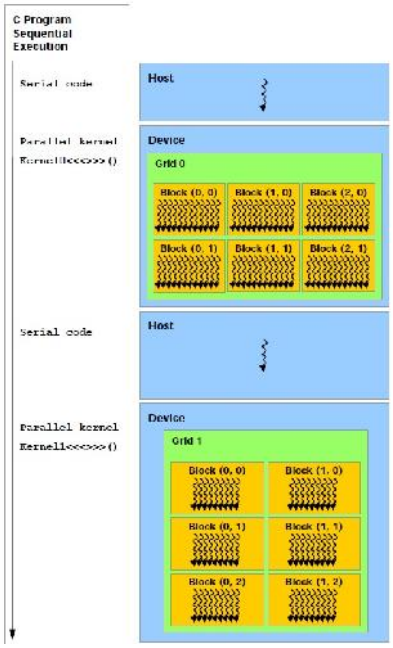
memoria también se accede a través de cache. Esta memoria tiene un límite de capacidad de 64 KB y debe ser inicializado por el host. El mejor desempeño de esta memoria se da cuando todos los threads de un warp leen de la misma dirección de memoria.

Modelo de Programación

El modelo de programación de CUDA está compuesto por dos características para aprovechar el poder de la GPU, el primero es una manera de organizar los threads en la GPU por una estructura de jerarquía y el segundo una manera de acceder a la memoria de la GPU por una estructura de jerarquía. Como se afirmó arriba, CUDA es computación heterogenia donde el *HOST* es la CPU y su memoria y el *DEVICE* es la GPU y su memoria. Cuando se lanza un kernel se ejecuta en la GPU, pero el control se devuelve al *HOST* de inmediato así liberando la CPU para que realice tareas adicionales. El flujo de un típico programa en CUDA sigue este orden y se evidencia en la figura 33.

1. Copiar datos de la memoria de la CPU a la memoria de la GPU.
2. Lazar los kernels para ejecución con los datos almacenados en la GPU.
3. Copiar datos de la memoria de la GPU a la memoria de la CPU.

Figura 33: Modelo de Programación



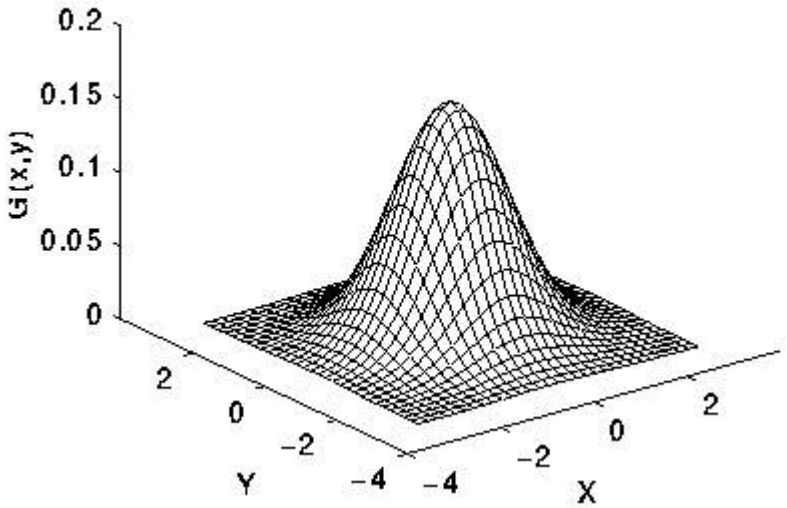
Fuente: NVIDIA CUDA™ Programming Guide 2.3.1 Capitulo 2 Pg. 13

3.2.11. Filtro Gaussiano

El filtro gaussiano en un tipo de convoluciòn para suavizar imágenes lo que tiene como resultado en remover detalle y eliminar ruido como el de puntos blancos y negros llamado *salt and pepper*. La imagen se pasa por un filtro pasa bajos el cual retiene los valores de baja frecuencia y elimina los valores de frecuencias altas. Los filtros de suavizado se aplican como un pre proceso para otras operaciones de imagen como, por ejemplo, a un algoritmo de detección de bordes ya que este algoritmo es sensible a ruido. También, este filtro de suavizado se pude aplicar una y otra vez hasta llegar al resultado deseado. Una de las desventajas de este filtro es la perdida de detalle o nitidez en la imagen y se debe tener en cuenta el *tradeoff* de la cantidad de ruido que se va a eliminar y la perdida de detalle. Este filtro es de dominio espacial, es decir se trabaja directamente con los pixeles de la imagen y su único parámetro es la desviación estándar de la distribución normal[42][43]. La siguiente formula describe la función gaussiana en dos dimensiones con media cero y su representación gráfica en la figura 34.

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figura 34: Distribución de forma gaussiana con media y desviación cero en 2-D



Fuente: Image Processing with CUDA. URL: *http://digitalscholarship.unlv.edu/*

3.2.12. Filtro Gradiente Sobel

La detección de bordes es una de las técnicas más utilizadas para la extracción de características en una imagen y disminuye la cantidad de datos para un proceso

posterior[43]. El operador Sobel se describe como una diferenciación discreta que calcula una aproximación del gradiente de la función de intensidad de la imagen[44]. El gradiente de una imagen provee dos tipos de información. La primera es la magnitud del gradiente que mide que tan rápido la imagen está cambiando, y segundo la dirección en la que está cambiando. Dado lo anterior, un borde está definido como una discontinuidad de de intesidad en una imagen de grises o cuando el gradiente de la imagen tiene una variación abrupta[9].

El operador Sobel es basado en la convolución de la imagen con un filtro pequeño en direcciones horizontales y verticales.

El operador sobel se define como:

$$S_1 = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad S_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Y si se tiene que I es la imagen se tiene la siguiente operación donde (*) es una convolución en 2D.

$$Gx = S_1 * I$$
$$Gy = S_2 * I$$

La imagen gradiente resultante es la combinación de Gx y Gy. Cada pixel de $G(x, Y)$ se puede ser calculado tomando la magnitud de Gx y Gy como se muestra en la siguiente ecuación:

$$G(x, y) = \sqrt{Gx^2 + Gy^2}$$

Finalizando se puede calcular la dirección del gradiente con la siguiente ecuación:

$$\theta = actan\frac{Gy}{Gx}$$

La idea de utilizar el número 2 en el coeficiente del centro es para darle más importancia al pixel del centro. A su vez, las operaciones Gx y Gy son lineales porque involucra derivadas pero al hallar la magnitud se vuelve no lineal ya que operaciones como de raíz cuadrada y valor absoluto no son lineales[45].

3.2.13. Reconstrucción Morfológica H-MIN

La reconstrucción es una transformación morfológica que involucra dos imágenes y un elemento estructurante al lugar de una imagen y un elemento estructurante[46]. Esta reconstrucción es conocida relativamente para imagen binarias, donde simplemente extrae componentes conectados que están marcados por otra imagen. Sin embargo, la reconstrucción también se puede definir para imágenes a escala de grises que resulta interesante para tareas de segmentación y extracción de características. Sorprendentemente, la reconstrucción morfológica a escala de grises ha traído poca atención en la comunidad de análisis de imagenes.[47].

La reconstrucción morfológica en escala de grises se puede obtener por sucesivas dilataciones geodésicas hasta que la imagen se estabilice o no haya más cambios. Como se mencionó anteriormente se utiliza una imagen marcadora y una imagen de mascara donde las dos imágenes deben tener el mismo tamaño y la imagen marcador debe tener valores menores o iguales a la imagen mascara[9].

La notación de la dilatación de una imagen M con un elemento estructurante está dada por:

$$\delta(M) = M \oplus B$$

Para una dilatación geodésica de tamaño 1 en la imagen marcador M con respecto a la imagen mascara I se define como:

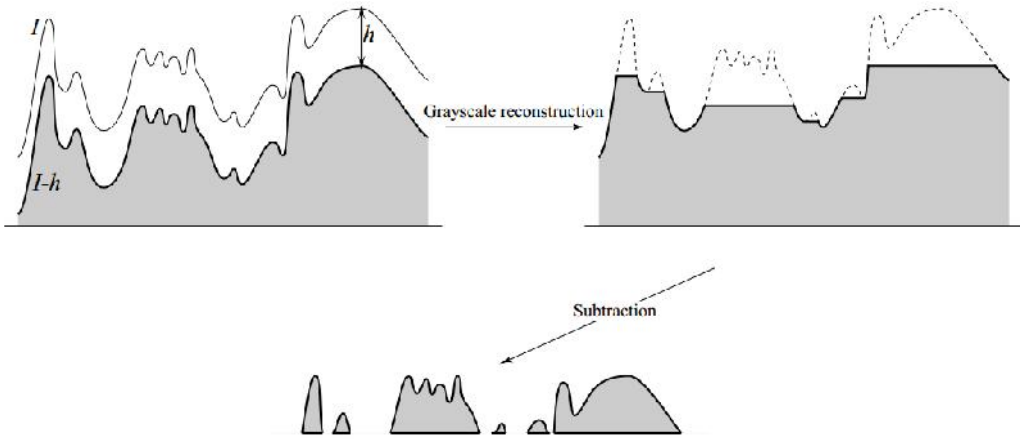
$$\delta_I^{(1)}(M) = (M \oplus B) \wedge I$$

El operador $\wedge$ denota el mínimo de intersección entre la imagen de marcador dilatada y la imagen de mascara[9].

La reconstrucción es un método eficiente para extraer máxima y mínima regional de imágenes en escala de grises. Un máximo regional se puede definir como componentes conectados de píxeles con un valor de intensidad constante, *t*, cuyos píxeles

de límite externo tienen un valor menor que *t*.La figura 35 muestra la reconstrucción morfológica h-máxima la cual se asigna una imagen marcador M, y a la imagen mascara I se le resta un valor h el cual es el atributo. En la figura 35 se puede observar el resultado de la reconstrucción morfológica h-máxima en una señal 1D.

Figura 35: Reconstrucción morfológica extrayendo cumbres de una señal 1D



Fuente: VINCENT

La ecuación de h-máxima esta dada por:

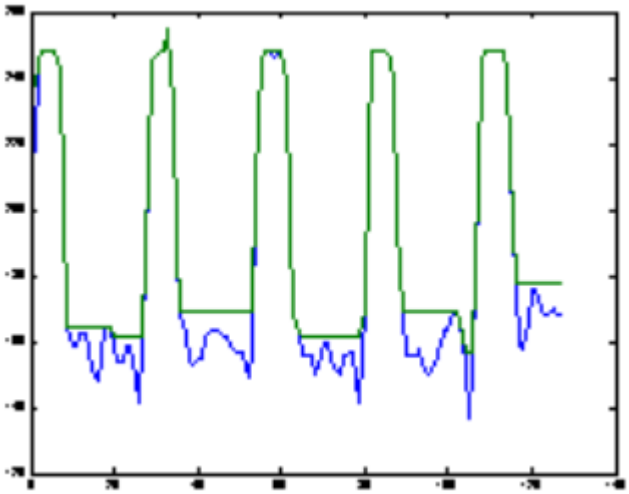
$$HMAX_{h,D}(I) = I\Delta(I - h)$$

Donde, D es la conectividad de pixeles, h es el atributo de altura, I es la imagen original y por ultimo Δ representa la reconstrucción morfológica. Por otro lado, mínima regional se puede definir como componentes conectados de píxeles con un valor de intensidad constante, y cuyos píxeles de límite externo tienen todos un valor más alto. La ecuación de h-mínima esta dada por:

$$HMIN_{h,D}(I) = (I^c\Delta_D(I - h)^c)^c$$

Donde c caracteriza una imagen invertida. La figura 36 representa el resultado del procesamiento $H - MIN$ para 1D[9].

Figura 36: Señal azul de entrada y resultado señal verde reconstrucción H-MIN



Fuente: BERNARDES

La finalidad de la reconstrucción H-MIN es eliminar los mínimos regionales de una imagen.

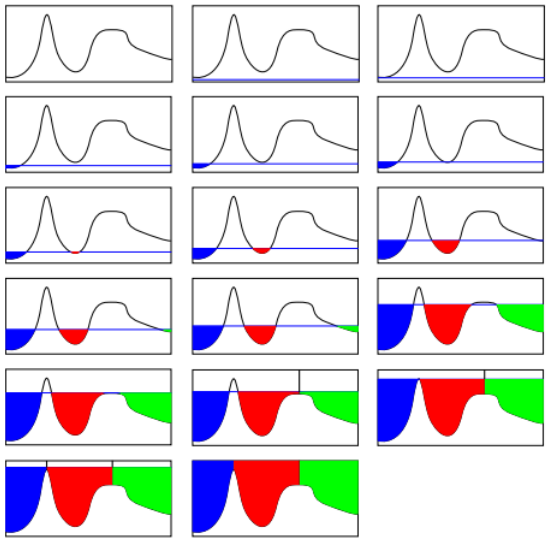
3.2.14. Transformada Watershed

La transformada watershed propone un enfoque morfológico para el problema de segmentación de imágenes, interpretando estas como superficies, donde cada pixel corresponde a una posición y los niveles de gris determinan las altitudes. A partir de esta noción, se desea entonces identificar cuencas hidrográficas, definidas por mínimos regionales y sus regiones de dominio [48]. Según Bernardes[9], la transformada watershed es ampliamente utilizada para la segmentación en visión computacional.

La transformada watershed trata de encontrar los puntos en una superficie donde una gota de agua puede escurrir en dos mínimos regionales diferentes. También, se puede describir como el nivel del agua es elevado a través de una superficie, inundando a partir de los mínimos regionales, y en los puntos donde aguas provenientes se mínimos diferentes se tocan se alza una barrera, que constituye la línea de división de cuencas[48].

La figura 37 muestra el relleno y la división por las barreras en cuando aguas de dos mínimos regionales se tocan.

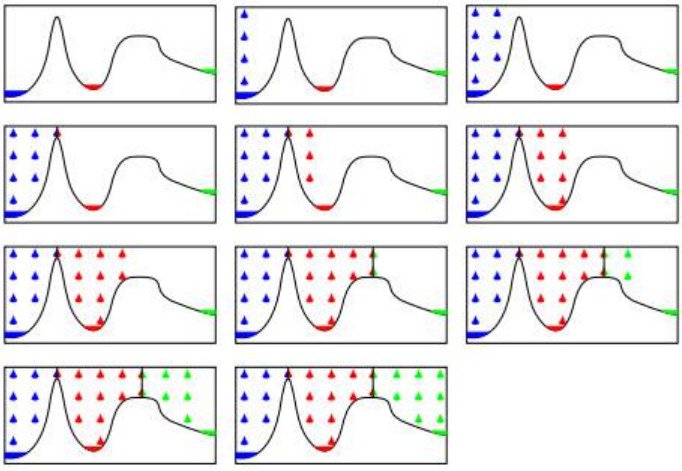
Figura 37: Ejemple de inundación de una superficie



Fuente: KORBES

Por otra parte, en la figura 38 se puede observar otra forma de interpretar la transformada watershed por gotas de agua. Como se puede observar una gota de agua escurre en dos mínimos regionales así creando la barrera de división.

Figura 38: Ejemple de inundación por gota de agua



Fuente: KORBES

Un problema de la transformada watershed es la sobre segmentación generando muchas más regiones de lo esperado. Para reducir este problema, se puede aplicar una reconstrucción morfológica para eliminar los mínimos regionales y así controlar

la segmentación.

3.3. Control Lógico Difuso

El concepto de lógica difusa está asociado a como las personas perciben el medio, por ejemplo, ideas relacionadas con la temperatura de una bebida, el tiempo de cocción de un alimento, la velocidad con la que se mueve un objeto, de forma cotidiana son formuladas de manera ambigua y depende quien percibe estos fenómenos. Una bebida puede estar fría o caliente, o un alimento puede estar crudo o cocinado, se dice que estas afirmaciones acerca de una variable son ambiguas porque frio o caliente son afirmaciones del observador, y pueden variar de un observador a otro. Uno se puede preguntar cuando algo se desplaza de forma lenta o rápida, a cuanta velocidad declaramos la lentitud o la rapidez, esto solo dependerá del observador.

Los conjuntos difusos definen justamente estas ambigüedades, y son una extensión de la teoría clásica de conjuntos, donde un elemento pertenece o no a un conjunto, tal elemento tiene solo dos posibilidades, pertenecer o no, un elemento bi-valuado y no se definen ambigüedades. Con conjuntos difusos se intenta modelar la ambigüedad con la que se percibe una variable. Los conjuntos difusos son la base para la lógica difusa. Con los conjuntos se realizan afirmaciones lógicas del tipo si-entonces, definiéndose estas con lógica difusa. Este tema es propio de la inteligencia artificial, donde se intenta emular el pensamiento humano.

Gracias a que Lotfy A. Zadeh [49] desarrollo el concepto de lógica difusa, se ha trabajado en este tema, el principal centro de desarrollo es Japón, donde sus investigadores la han aplicado a muy diversos sistemas, principalmente electrodomésticos, sistemas más recientes están vinculados con la industria, la medicina y la actividad espacial. Muchas publicaciones y libros se han escrito de este tema, pero aún queda mucho por explorar.

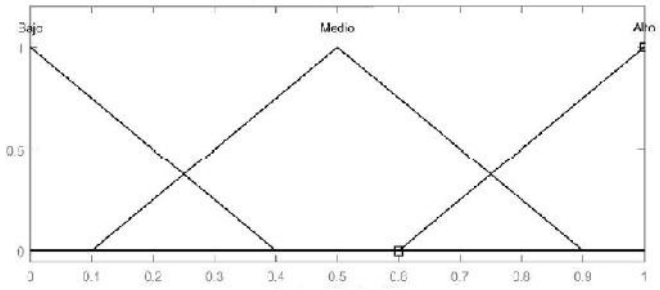
La toma de decisión a partir de información que no especifica también es un procedimiento cotidiano, esto es lo que se intenta emular con la lógica difusa a partir de: la observación del entorno, la formulación de reglas lógicas y de los mecanismos de toma de decisión. Los conceptos básicos de lógica difusa que se aplican en control, tales son conjuntos difusos, funciones de membresía, operaciones borrosas, reglas, inferencia, defusificación y los pasos para la toma de decisión.

3.3.1. Conjuntos Borrosos

Los conjuntos clásicos, tiene limitaciones, se define un universo de discurso que contiene a conjuntos cuyos bordes están definidos, un elemento puede o no perte-

necer a cierto conjunto, algo es verdadero o falso, no se definen situaciones intermedias. Los conjuntos borrosos son una extensión de los clásicos, donde se añade una función de pertenencia, definida esta como un número real entre 0 y 1. Así se introduce el concepto de conjunto o subconjunto borroso y se lo asocia a un determinado valor lingüístico, definido por una palabra o etiqueta lingüística, donde esta es el nombre del conjunto o subconjunto. Por cada conjunto se define una función de pertenencia o membresía denominada $\mu_{A(x)}$, indica el grado en que la variable x está incluida en el concepto representado por la etiqueta A $\Big(0 \leq \mu_{A(x)} \leq 1\Big)$, si esta función toma el valor 0 significa que tal valor de x no está incluido en A y si toma valor 1 el correspondiente valor de x está totalmente incluido en A. En la figura 39 se podrá apreciar un ejemplo donde el conjunto altura (con variable x) está dividido en 3 subconjuntos $\{Bajo, \quad Medio, \quad Alto\}$ con sus respectivas funciones de membresía $\Big\{\mu_{Bajo(x)}, \quad \mu_{Medio(x)}, \quad \mu_{Alto(x)}\Big\}$.

Figura 39: Subconjunto borroso para conjunto altura.



Fuente: Propia del autor

Sea X una colección de objetos, expresados en forma genérica por x. Entonces, un conjunto difuso A en X, se define como un conjunto de pares ordenados.

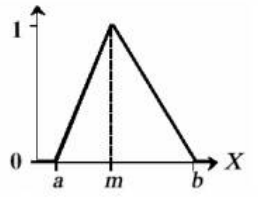
$$A = \Big\{\Big(x, \mu_{A(x)}\Big)/x \in X\Big\}$$

3.3.2. Funciones de Membresía

Las funciones de membresía representan el grado de pertenencia de un elemento a un subconjunto definido por una etiqueta. Existe una gran variedad de formas para las funciones de membresía, las más comunes son del tipo trapezoidal y triangular.

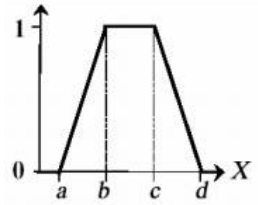
Forma triangular

$$A(x) = \begin{cases} 0 & si \quad x \leq a \\ (x-a)/(m-a) & si \quad x \in (a,m] \\ (b-x)/(b-m) & si \quad x \in (m,b) \\ 1 & si \quad x \geq b \end{cases}$$



Forma Trapezoidal

$$A(x) = \begin{cases} 0 & si \quad (x \leq a) o (x \geq d) \\ (x-a)/(b-a) & si \quad x \in (a,b] \\ 1 & si \quad x \geq (b,c) \\ (d-x)/(d-c) & si \quad x \in (b,d) \end{cases}$$



3.3.3. Operaciones Borrosas

A los subconjuntos se les puede aplicar determinados operadores o bien se puede realizar operaciones entre ellos. Al aplicar un operador sobre un solo conjunto se obtendrá otro conjunto, lo mismo sucede cuando se realiza una operación entre conjuntos. Las operaciones lógicas se utilizan en controladores y modelos difusos, son necesarias en la evaluación del antecedente de reglas.

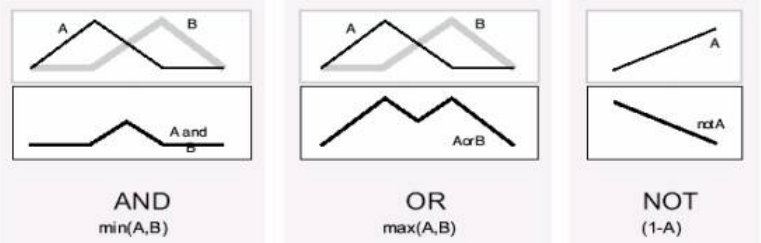
Estas tres operaciones son las más básicas a realizar sobre conjuntos, son complemento, unión e intersección. Sean las etiquetas A y B las que identifican a dos conjuntos borrosos asociados a una variable lingüística x, las operaciones se definen como:

- Complemento $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$
- Unión. Operador lógico OR de Zadeh (max) $\mu_{A \cup B}(x) = max[\mu_A(x), \quad \mu_B(x)]$
- Intersección. Operador lógico AND de Zadeh (min) $\mu_{A \cap B}(x) = min[\mu_A(x), \quad \mu_B(x)]$

Hay muchas definiciones para las operaciones lógicas, algunas otras definiciones que normalmente también se utilizan son:

- Operador logico AND del producto $\mu_{A \cap B}(x) = \mu_A(x) \times \mu_B(x)$
- Operador logico OR de Lukasiewicz $\mu_{A \cup B}(x) = max[\mu_A(x) + \mu_B(x), 1]$

Figura 40: Operaciones borrosas

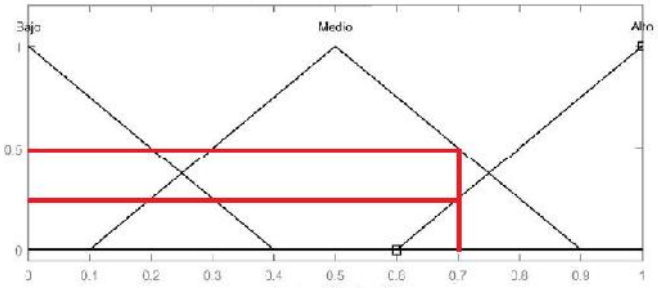


Fuente: http://www.biblioteca.udep.edu.pe

3.3.4. Fuzzificación

El control difuso siempre involucra este proceso de fuzzificación, esta operación se realiza en todo instante de tiempo, es la puerta de entrada al sistema de inferencia difusa. Es un procedimiento matemático en el que se convierte un elemento del universo de discurso en un valor en cada función de membresía las cuales pertenece.

Figura 41: Fuzzificación de una variable



Fuente: Propia del autor

Al observar la figura 41 se comprende mejor lo descrito anteriormente, de lo cual se puede concluir los siguientes datos:
$\mu_{Bajo}\,(0{,}7) = 0$
$\mu_{Medio}\,(0{,}7) = 0{,}5$
$\mu_{Alto}\,(0{,}7) = 0{,}25$

El valor de altura $0{,}7$ pertenece a dos conjuntos con distintos grados en cada uno.

3.3.5. Reglas Borrosas

Los controladores difusos usan reglas, estas combinan uno o más conjuntos borrosos de entrada llamados antecedentes o premisas y le asocian un conjunto borroso de salida llamado consecuente o consecuencia. Involucran a conjuntos difusos, lógica difusa e inferencia difusa. A estas reglas se les llama reglas borrosas o difusas o fuzzy rules. Son afirmaciones de tipo SI-ENTONCES. Los conjuntos borrosos del antecedente se asocian mediante operaciones lógicas borrosas AND, OR, etc.

Las reglas borrosas son proposiciones que permiten expresar el conocimiento que se dispone sobre la relación entre antecedentes y consecuentes. Para expresar este conocimiento de manera completa normalmente se precisan varias reglas, que se agrupan formando lo que se conoce como base de reglas, es decir, la edición de esta base determina cual será el comportamiento del controlador difuso y es aquí donde se emula el conocimiento o experiencia del operario y la correspondiente estrategia de control.

La base de las reglas suele representarse por tablas. Esta es clara en el caso de dos variables de entrada y una de salida. En la medida que la cantidad de variables lingüísticas crece, también lo hará la tabla, y más difícil se hará su edición. Junto a cada regla puede estar asociado un valor entre cero y uno que pesa a tal regla, esto puede ser importante cuando una regla tiene menor fuerza que otras de la base de reglas.

Existe una gran variedad de tipos de reglas, dos grandes grupos son los que en general se emplean, las reglas difusas de Mandami y las reglas difusas de Takagi-Sugeno. La estructura de las reglas es la misma tanto para controladores como para modelos, simplemente cambiara las variables implementadas.

Reglas difusas de Mandami

$$IF \quad x_1 isA \quad AND \quad x_2 isB \quad AND \quad x_3 isC \quad THEN \quad u_1 isD, \quad u_2 isE$$

Donde x_1, x_2 y x_3 son las variables de entrada (por ejemplo, error, derivada del error e integral del error), A, B y C son funciones de membresía de entrada (por ejemplo, alto, medio y bajo), u_1 y u_2 son las acciones de control (por ejemplo, apertura de válvulas) en sentido genérico son todavía variables lingüísticas, D y E son las funciones de membresía de la salida, y AND es un operador lógico difuso, podría ser otro. La primera parte de la sentencia "$IF \quad x_1 isA \quad AND \quad x_2 isB \quad AND \quad x_3 isC$" es el antecedente y la restante es el consecuente.

VENTAJAS:

- Es intuitivo.
- Tiene una amplia aceptación.
- Está bien adaptado a la incorporación de conocimiento y experiencia.

Reglas difusas de Takagi-Sugeno

$$IF \quad x_1 isA \quad AND \quad x_2 isB \quad AND \quad x_3 isC \quad THEN \quad u_1 = f(x_1, x_2, x_3), \quad u_2 = g(x_1, x_2, x_3)$$

En principio es posible emplear $f()$ y $g()$ como funciones no lineales, pero la elección de tal función puede ser muy compleja, por lo tanto en general se emplean funciones lineales.

VENTAJAS

- Es computacionalmente eficiente.
- Trabaja bien con técnicas lineales.
- Trabaja bien con técnicas de optimización y control adaptable.
- Tiene garantizada una superficie de control continua.
- Está bien adaptado al análisis matemático.

3.3.6. Agregado

Cuando se evalúan las reglas se obtienen tantos conjuntos difusos como reglas existan, para defusificar es necesario agrupar estos conjuntos, a esta etapa se le llama **agregado** y existen varios criterios para realizar este paso. Un criterio muy empleado es el agrupar los conjuntos inferidos mediante la operación max.

3.3.7. Defusificción

La defusificación es un proceso matemático usado para convertir un conjunto difuso en un número real. El sistema de inferencia difusa obtiene una conclusión a partir de la información de la entrada, pero es en términos difusos. Esta conclusión o salida difusa es obtenida por la etapa de inferencia borrosa, esta genera un conjunto borroso pero el dato de salida del sistema debe ser un número real y debe ser representativo de todo el conjunto obtenido en la etapa de agregado, es por eso que existen diferentes métodos de defusificación y arrojan resultados distintos, el más común y ampliamente usado es centroide. Con el método de defusificación del centroide se transforma la salida difusa en un número real el cual es la coordenada (x) del centro de gravedad de tal conjunto difuso de salida.

$$y_d = \frac{\int_s y\mu_Y(y)dy}{\int_s \mu_Y(y)dy}$$

La ecuación anterior demuestra la ecuación de defusificación por centro de gravedad donde μ_Y es la función de pertenencia del conjunto de salida Y, cuya variable de salida es y. S es el dominio o rango de integración. Este método en realidad trae una carga computacional importante, por lo que se emplean en general otros esquemas con mejor carga.

Uno de los defusificadores más usados es el centro de área (COA, center of area) también llamado de altura, el centro de gravedad es aproximado por el centro de gravedad de un arreglo de masas puntuales, las cuales son el centro de gravedad de cada conjunto de salida correspondiente a cada regla, con masa igual al grado de pertenencia en ese punto de su centro de gravedad. Si se le llama δ_l al centro de gravedad del conjunto difuso de salida B_l de la $l-esima$ regla, el centro de gravedad queda determinado por:

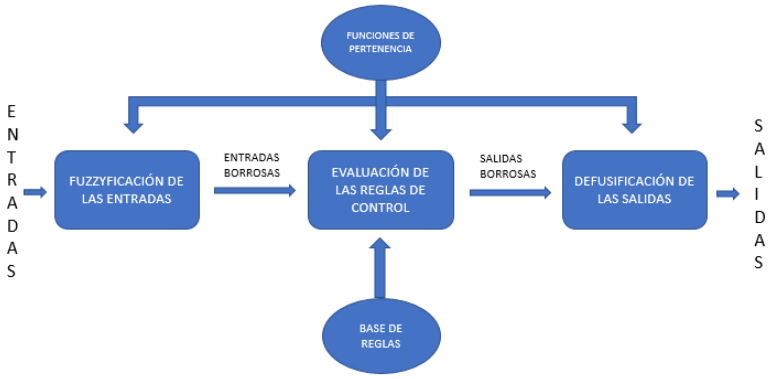
$$y_d = \frac{\sum_{l=1}^{R} \delta_l \mu_{B1(\delta l)}}{\sum_{l=1}^{R} \mu_{B1(\delta l)}}$$

Donde R es el número de reglas. El concepto de centro de gravedad es en muchos casos el punto de partida para la obtención de distintos métodos de defusificación [50]. Tanto la fuzzificación como la defusificación son el nexo del sistema difuso con el mundo real.

3.3.8. Toma de decisión

Ahora bien, ya presentados los conceptos básicos de un sistema de inferencia difusa o de toma de decisión. Se verá ahora de manera resumida y en forma gráfica en la figura 46 los pasos que son llevados a cabo para la toma de decisión en este sistema de inferencia difusa.

Figura 42: Diagrama de pasos para toma de decisión en un sistema de inferencia difusa



Fuente: Propia del autor

3.3.9. Lógica difusa y sistemas de control

La incorporación de lógica difusa a los sistemas de control da lugar a lo que se llama sistemas de control difuso. Dentro de los sistemas de control se encuentran grandes áreas, el modelado o identificación y el control propiamente dicho o control directo. El objetivo de este es muy simple, se trata de determinar de manera lógica que se debe hacer para lograr los objetivos de control de mejor manera posible a partir de una base de conocimiento proporcionada por un operador humano, sin esta base no es posible desarrollar una aplicación y que esta funcione de manera correcta.

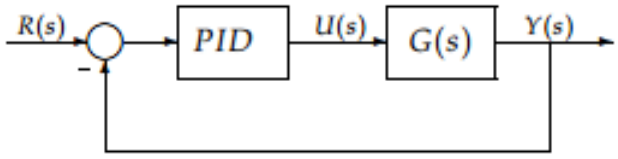
Se necesita el conocimiento y experiencia de un operador humano para construir un controlador que emule el comportamiento de tal persona. Comparado con el control tradicional, el control difuso tiene dos ventajas prácticas, una es que el modelo matemático del proceso a controlar no es requerido y otra es que se obtiene un controlador no lineal desarrollado empíricamente sin complicaciones matemáticas [51].

3.4. Controlador PID

Una de las técnicas más utilizadas en el mundo de los controladores, conciste utilizar una transformación no lineal para obtener un sistema lineal. Sobre las cuales se utilizan técnicas clásicas de control como los PID. Un sistema cerrado es aquel sistema realimentado el cual la salida de sistema se vuelve a introducir adicionándola a la entrada.

La idea del control en lazo cerrado es emplear precisamente esta característica para conseguir que la respuesta del sistema sea la que se desee. El simple hecho de realimentar un sistema, no es suficiente para conseguir esto. La razón es, que normalmente los parámetros de la función de transferencia de un sistema vienen dados por la dinámica del sistema. Una solución a esto, es añadir al sistema original $G(t)$ un segundo sistema a la entrada llamado $C(t)$, cuyos parámetros se pueden ajustar a voluntad. Este segundo sistema recibe el nombre de controlador y al sistema original se le da nombre de planta. De este modo, el controlador recibe como estrada la diferencia entre un valor de referencia $r(t)$ que es el valor que se desea obtener a la salida de nuestra planta y la salida real de la planta $y(t)$. La respuesta del controlador tendrá que ser tal, que al cabo del tiempo consiga hacer que $r(t) - y(t) = 0$ [52].

Figura 43: Modelo de control en lazo cerrado PID



Fuente: http://www.eng.newcastle.edu.au/ jhb519/teaching/caut1/Apuntes/PID.pdf

La entrada al controlador $e(t)$ recibe el nombre de señal de error ya que representa la diferencia entre el valor de referencia y la salida real de la planta. La salida del controlador recibe el nombre de señal de control $c(t)$, siendo esta señal la nueva entrada de la planta. La cuestión es como diseñar el controlador para conseguir la señal deseada, volver la señal de error en cero. La solución más sencilla posible es considerar un control cuya salida sea directamente proporcional al error $(controlador \quad Proporcional)$, es decir, si la salida de la planta es menor que la señal de referencia, la señal de control será positiva y proporcional a $r(t) - y(t)$, si por el contrario fuera mayor, la señal de referencia será ahora negativa y proporcional a $r(t) - y(t)$. A este tipo de controlador se le llama proporcional. El primero de ellos, es cuando la planta alcanza valor consigna, la señal de control se hace cero. Para muchos sistemas esto no es una entrada valida, lo que acabaría provocando una oscilación en la entrada del sistema. Pero además, para muchos sistemas, ni siquiera es capaz de hacer que la planta alcance la respuesta deseada.

Una solución para este problema es añadir un segundo término proporcional a la integral del error, de alguna manera, este segundo término actúa como una $memoria$ que lleva la historia previa del error. Cuando se añade al controlador, evita que este alcance un valor de equilibrio para un error dado, como en el proporcional, este tipo

de controlador es llamado PI.

Si al controlador PI descrito anteriormente se le agrega un término que sea proporcional a la derivada del error, el resultado es un controlador PID completo. Un término proporcional a la derivada del error significa que, si este crece con el tiempo, si derivada será positiva y tanto mayor cuanto más rápido crezca. Si por el contrario decrece, la derivada será negativa, y de nuevo tanto más grande en valor absoluto cuanto más rápido decrezca. Esta propiedad, que hace el termino derivativo de alguna manera se anticipe al valor del error, puesto que si acción depende del ritmo al que este está variando[53].

Tipo de controlador	Ecuación
Proporcional	$C_p(s) = K_p$
Integral	$C_i(s) = \frac{K_i}{s}$
Proporcional-integral	$C_{PI}(s) = K_p\left(1 + \frac{1}{T_i s}\right)$
Proporcional-derivativo	$C_{PD}(s) = K_p + sK_pT_d$
Proporcional-integral-derivativo	$C_{PID}(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$

3.4.1. Métodos de diseño para controladores PID

Los métodos para el desarrollo de controladores PID son muy variados, pero en este apartado se hablará de dos tipos de métodos muy comunes como lo son Ziegler-Nichols y Lugar de las Raíces.

■ **Reglas de sintonía de Ziegler-Nichols**

El controlador PID recibe una señal de entrada (generalmente es el error) y proporciona una salida (acción de control, $u(t)$).

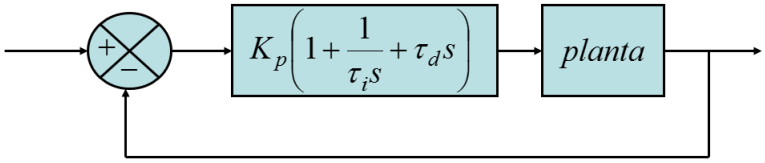
$$u(t) = K_p\left(e(t) + \frac{1}{\tau_i}\int_{-\infty}^{t} e(t)dt + \tau_d\frac{de(t)}{dt}\right)$$

Entonces, la función de transferencia del controlador PID es:

$$G_c(s) = K_p\left(1 + \frac{1}{\tau_i s} + \tau_d s\right)$$

donde K_p es la ganancia proporcional,τ_i el tiempo integral y τ_d es el tiempo derivativo. El esquema habitual de uso del controlador PID es:

Figura 44: Esquema de uso del controlador PID



Fuente: Propia del autor

Ziegler y Nichols propusieron una serie de reglas para afinar controladores PID con base a una respuesta experimental. Definieron dos métodos.

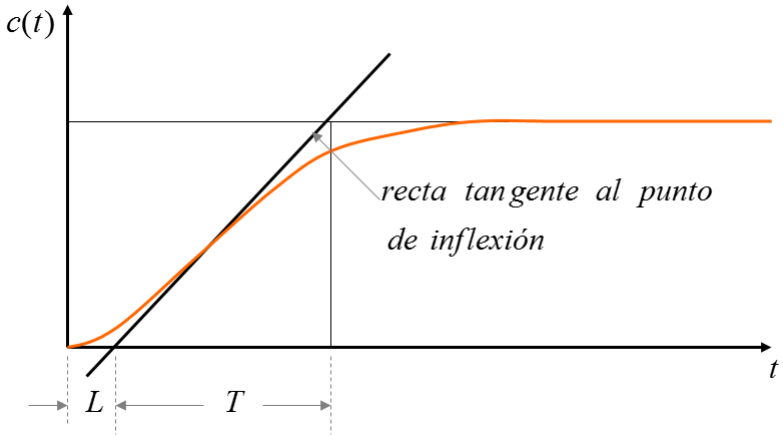
Primer Método

Se obtiene experimentalmente la respuesta de la planta a una entrada escalón y si la respuesta no tiene oscilaciones y además posee un retardo tal que se forma una “ese”, puede obtenerse los parámetros del controlador PID utilizando el primer método.

Esta respuesta se caracteriza con el tiempo de atraso L y la constante de tiempo T. Y se puede aproximar por un sistema de primero orden con atraso de transporte.

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1}$$

Figura 45: Curva experimental en forma de “ese”



Fuente: Propia del autor

Para obtener L y T, se traza una recta tangente al punto de inflexión de la respuesta, la intersección con el eje del tiempo y con el valor final de la amplitud forman

las distancias L y T.

Con L y T, se obtienen los parámetros del controlador PID utilizando la tabla.

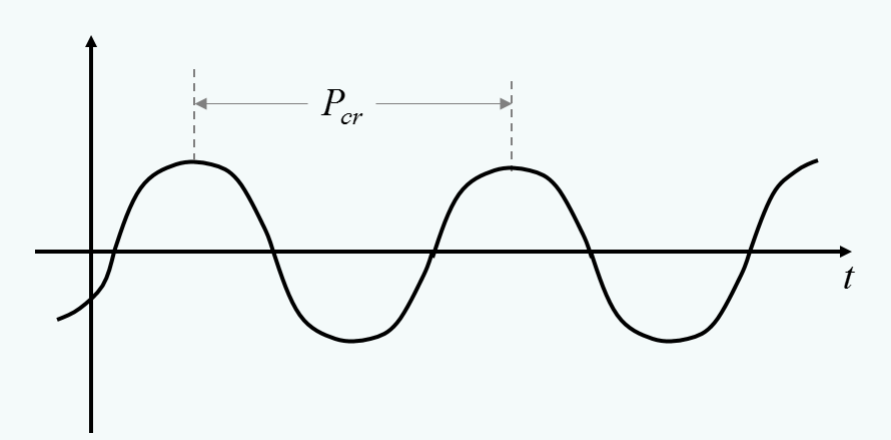
Tabla 3: Valores de sintonización, método uno

Tipo de controlador	K_p	τ_i	τ_d
P	$\frac{T}{L}$	∞	0
PI	$0{,}9\frac{T}{L}$	$\frac{L}{0{,}3}$	0
PID	$1{,}2\frac{T}{L}$	$2L$	$0{,}5L$

Segundo Método

Se utiliza para sistemas que pueden tener oscilaciones sostenidas. Primero se eliminan los efectos de la parte integral y derivativa. Después, utilizando solo la ganancia K_p, haga que el sistema tenga oscilaciones sostenidas. El valor de ganancia con que se logre esto se llama ganancia crítica K_{cr}, que corresponde a un periodo crítico P_{cr}.

Figura 46: Oscilación sostenida



Fuente: Propia del autor

Con los valores de K_{cr} y P_{cr} se calculan lo valores de los parámetros del controlador PID, utilizando la tabla.

Tabla 4: Valores de sintonización, método dos

Tipo de controlador	K_p	τ_i	τ_d
P	$0{,}5K_{cr}$	∞	0
PI	$0{,}45K_{cr}$	$\frac{1}{2}P_{cr}$	0
PID	$0{,}6K_{cr}$	$0{,}5P_{cr}$	$0{,}125P_{cr}$

■ Método de lugar de las raíces

Es el lugar geométrico de los polos y ceros de una función de transferencia que se varía la ganancia K en un determinado intervalo.

El método de lugar de raíces permite determinar la pocisión de los polos de la funcion de transferencia a lazo cerrado para un determinado valor de ganancia K a partir de la función de transferencia a lazo abierto.
La estrategia de diseño de controladores PID mediante el lugar de las raíces consiste en:

- Elegir un par de polos complejos adecuados para que en bucle cerrado sean los dominantes y definan una respuesta del sistema correcta (con poca sobreoscilación y un tiempo de establecimiento suficientemente pequeño).
- Elegir el tipo de controlador (puede ser P, PD, PI ó PID).
- Fijar los ceros del PID para que el lugar de las raíces pase por los polos anteriores.
- Obtener el valor de K_c (ganancia del controlador) para que los polos en bucle cerrado sean los deseados.
- Obtener el resto de polos y ceros en bucle cerrado y comprobar la respuesta del sistema resultante, volviendo al punto 1 si fuera necesario.

Para elegir el par de polos complejos conjugados que definirán la respuesta del sistema controlado se realiza la siguiente suposición:

Se supondrá que en bucle cerrado hay un par de polos complejos conjugados que son dominantes, es decir, el resto de polos son más rápidos que éstos.

Si la hipótesis es cierta, el comportamiento del sistema final es muy similar al de un sistema de segundo orden estándar. En este tipo de sistema, las características de la respuesta dependen de los polos o de la ecuación característica.

Las especificaciones dinámicas en el dominio del tiempo más empleadas son:

Sobreimpulso

$$M_p = e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}$$

Tiempo de asentamiento

$$t_s = \frac{4}{\zeta w_n}$$

Tiempo pico

$$t_p$$

Error en estado estacionario

$$e_e$$

Recordando que la ecuación de los polos dominantes, resultantes de una dinámica de segundo orden, esta dada por:

$$p_i = -\zeta w_n \pm w_n\sqrt{1-\zeta^2} \quad j = -\sigma \pm w_p j$$

Puede observarse de las ecuaciones anteriores que, el tiempo de establecimiento depende únicamente de la parte real de los polos y la sobreoscilación depende únicamente del amortiguamiento.

Lo habitual es que se fijen los polos de forma que la sobreoscilación del sistema en bucle cerrado sea entre el 4 y el 10 %, es decir, que la parte imaginaria tenga un valor similar o algo mayor que la parte real.

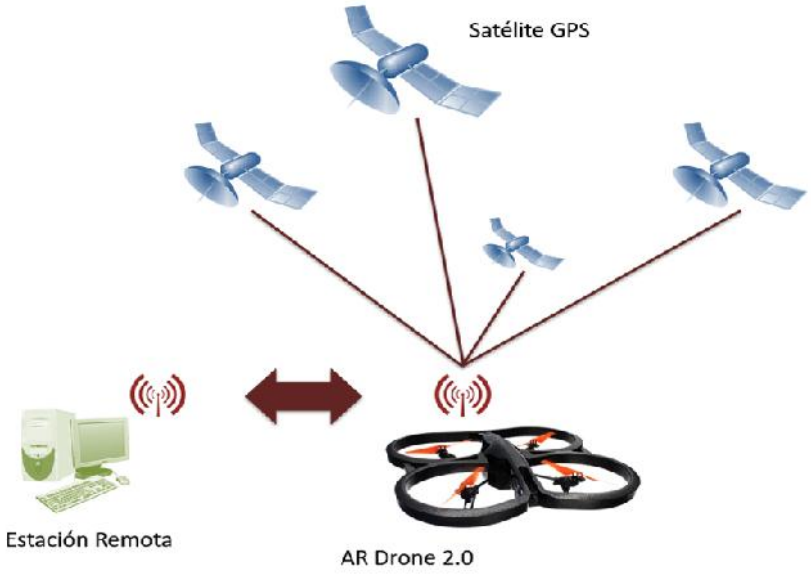
3.5. Comunicación

El AR Drone 2.0 puede ser controlado por cualquier dispositivo que soporte Wifi. Al momento de prender el AR Drone 2.0 crea su propia red WiFi con nombre **adrone2xxx** y asigna una dirección IP 192.168.1.1.

El control de AR.Drone se realiza a través de 3 servicios de comunicación principales. El primero, envía *AT commands* al puerto UDP 5556 para el control y configuración del AR Drone. Estos comandos se envían al AR Drone generalmente 30 veces por segundo y son fundamental para la experiencia del usuario. Por otra parte, la información del AR Drone como su estado, posición, velocidad de rotación de los motores , entre otros, es enviada al puerto UDP 5554. Esta información se llama *navdata* y se envía 15 veces por segundo en el modo demo y 200 veces en el modo completo(debug).

La transmision de video es enviada al puerto TCP 5555. Las imagenes de la transmision de video es codificada por la libreria FFMPEG. Por ultimo, un cuarto canal de comunicación, denominado puerto de control, se puede establecer en el puerto TCP 5559 para transferir datos críticos, por oposición a los otros datos que se pueden perder sin ningún efecto peligroso. Se utiliza para recuperar datos de configuración y para reconocer información importante como el envío de información de configuración[54]. El esquema de comunicacion se puede observar en la figura 47.

Figura 47: Comunicación

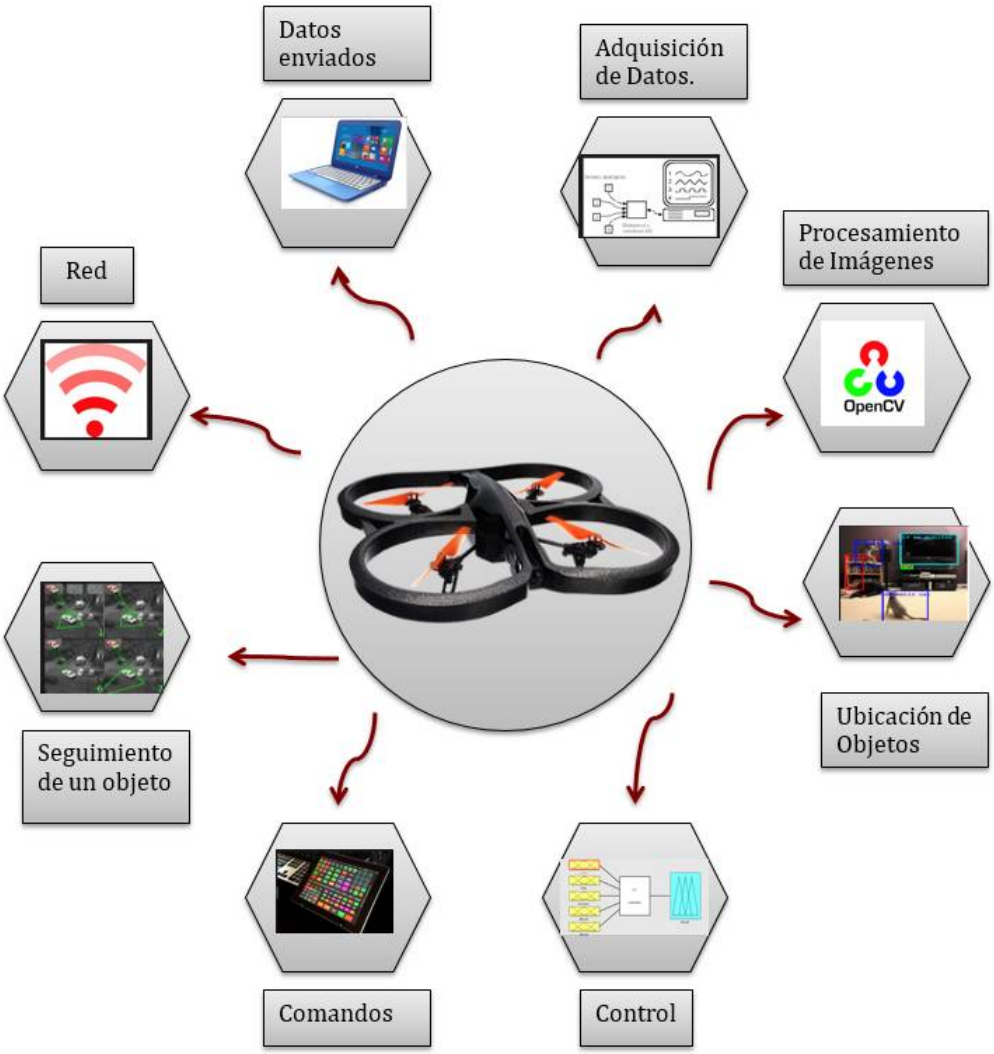


Fuente: Propia del autor

4. Integración Mecatrónica

En este capítulo se describe la integración mecatrónica del proyecto como se muestra en la figura 48. También se explicara la implementación de algoritmos en arquitectura como la conversión de una imagen RGB a escala de grises, un filtro gaussiano para suavizado y el filtro sobel para la detección de bordes.

Figura 48: Integración Mecatrónica



Fuente: Propia del autor

4.1. Diseño de Controladores

Para determinar un vuelo estable en el quadrotor se debe controlar cuatro variables las cuales tres son los ángulos Euler Pitch, Roll y Yaw y la última es la altura; para esto se diseñaron dos tipos de controladores, uno de tipo PID y otro difuso, para este las entradas el error y la derivada del error de cada una de estas variables, con esto se fijan las ganancias para así dar la potencia que los motores necesitan para que el quadrotor cumpla con la trayectoria establecida. Para la lógica difusa de control se ha utilizado Fuzzy Logic Toolbox de Simulink y se han creado los archivos Altura.fis, Pitch.fis, Roll.fis y Yaw.fis para el control de altura y los ángulos Euler respectivamente. Para los controladores PID son diseñados por los métodos de Ziegler-Nichols y lugar geométrico de las raíces

4.1.1. Controladores difusos

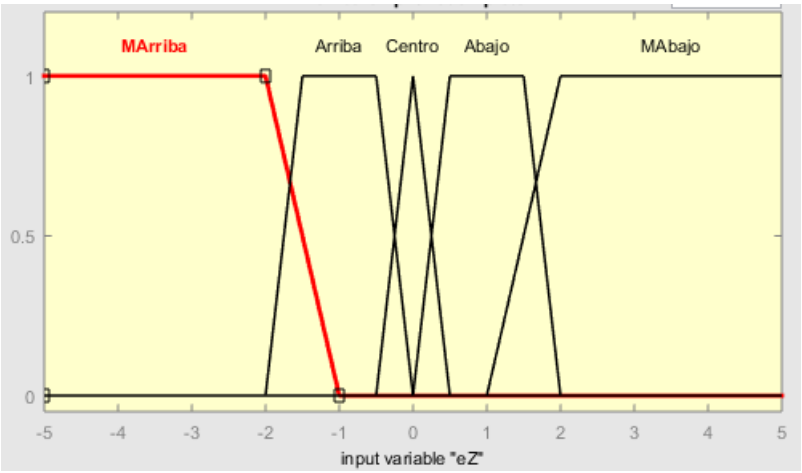
Todos los controladores difusos tienen los siguientes puntos en común:

- Se ha utilizado la estructura de Mandami como tipo de inferencia.
- Se ha utilizado el centro de gravedad como método de fuzzyficación.
- La operación borrosa empleada es de tipo AND.

■ **Control de altura**

Es el encargado de hallar la potencia para controlar la altura del quadrotor. Para determinar los parámetros de este primer controlador solamente es necesario disponer de las ecuaciones relativas al movimiento vertical, excluyendo las relativas a los movimientos rotatorios. En este caso se determinó que el punto de estabilización del quadrotor será en 5m.

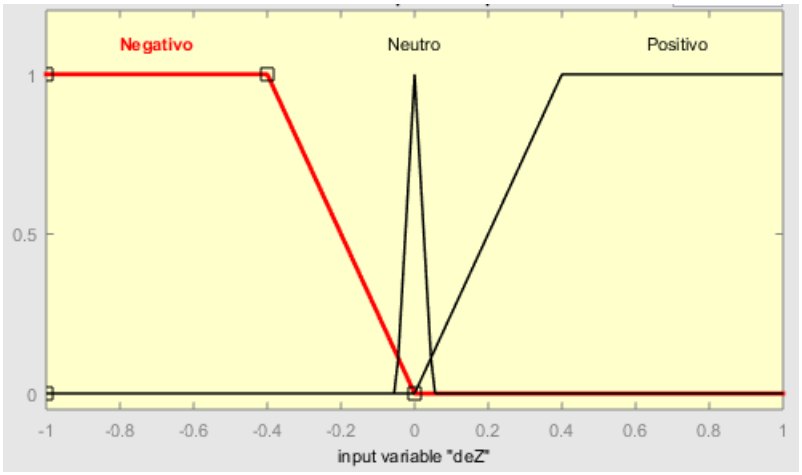
Figura 49: Funciones de pertenencia para error de altura **(eZ)**.



Fuente: Propia del autor

La derivada del error de altura o como se le llama deZ tiene un rango de $[-1, 1]$ dividiendo el universo de conjuntos borrosos en tres: Negativo, Neutro y Positivo.

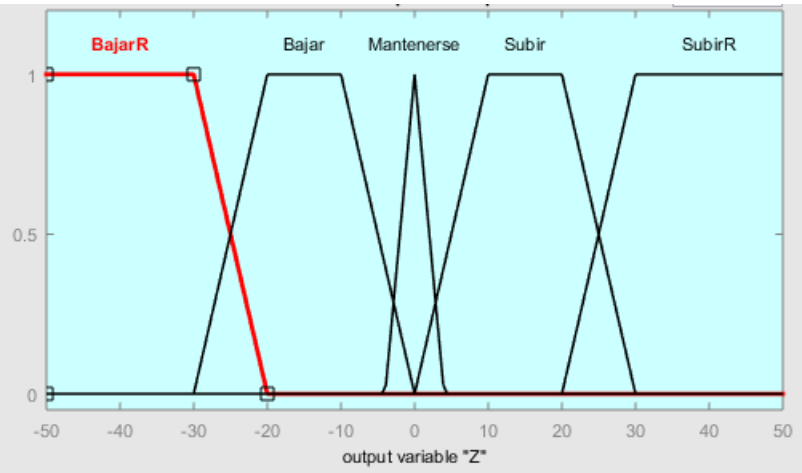
Figura 50: Funciones de pertenencia para derivada del error e altura **(deZ)**.



Fuente: Propia del autor

La variable de salida que hace referencia a la potencia del motor, para esto se ha utilizado cinco conjuntos borrosos para la salida de control: Bajar Rápido (BR), Bajar(B), Mantenerse (M), Subir (S) y Subir Rápido (SR).

Figura 51: Funciones de pertenencia para salida de control de altura **(Z)**.

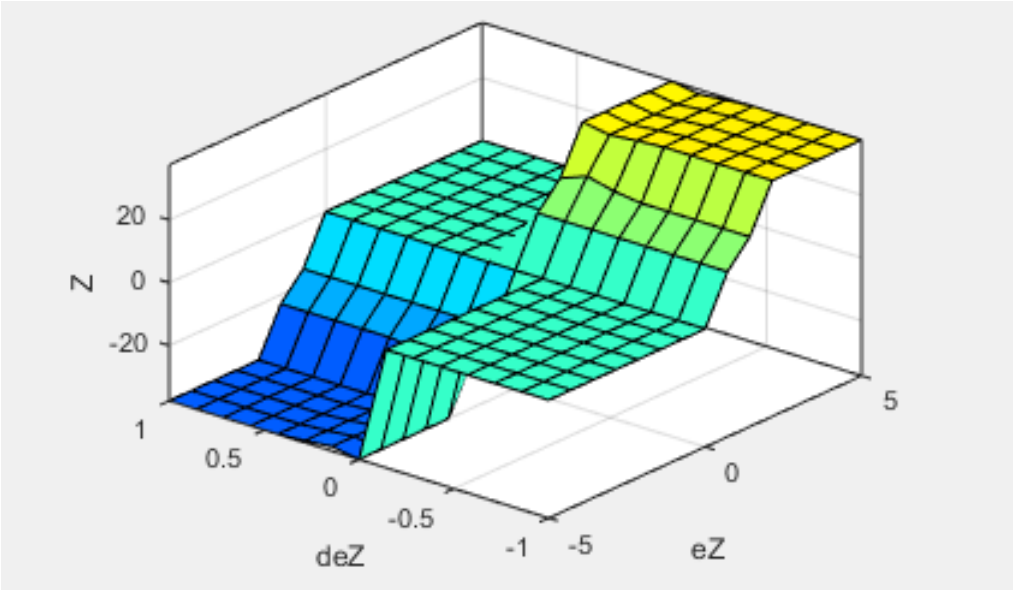


Fuente: Propia del autor

deZ / eZ	MA	A	C	Ab	MAb
Positivo	BR	B	B	S	SR
Neutro	BR	B	M	S	SR
Negativo	BR	B	S	S	SR

Reglas de inferencia de altura..

Figura 52: Gráfico de superficie generada por las composiciones de las reglas.



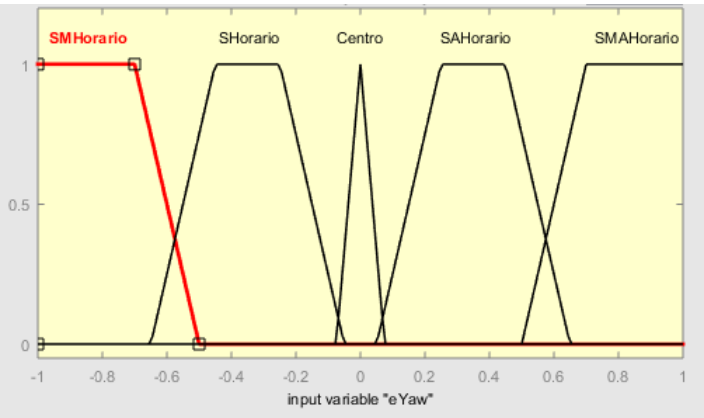
Fuente: Propia del autor

■ **Control de Yaw**

Para el control en Yaw es necesario utilizar las ecuaciones físicas relativas a el ángulo Yaw, también es necesario hacer las calibraciones oportunas equilibrando desajustes que pudieran afectar a la altura, a Pitch y Roll. El objetivo de este controlador es minimizar la variación del ángulo durante la ejecución de la trayectoria; las variables de entrada al controlador son el error en Yaw su respectiva derivada.

Los cinco conjuntos borrosos que dividen la variable $eYaw$ se han llamado: Muy Horario (MH), Horario (H), Centro (C), Anti horario (AH) y Muy Anti horario (MAH). En referencia a si el giro que se quiere realizar es en sentido horario o anti horario o se quiere mantener el rumbo. El rango de los valores para esta variable lingüística se muestra en la Figura53.

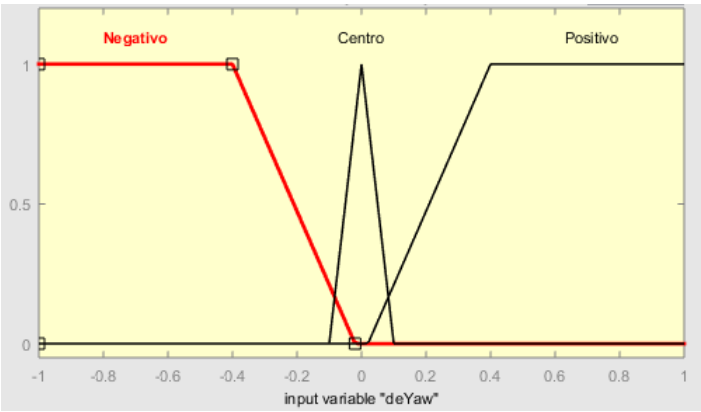
Figura 53: Funciones de pertenencia para error en Yaw **(eYaw)**.



Fuente: Propia del autor

La variable $deYaw$ se divide en tres conjuntos borrosos los cuales son llamados: Positivo, Negativo y Neutro, etiquetas lingüísticas que recogen si el valor de la derivada del error en Yaw es positiva, negativa o aproximada a cero, en donde el rango de esta variable es de $[-1, 1]$.

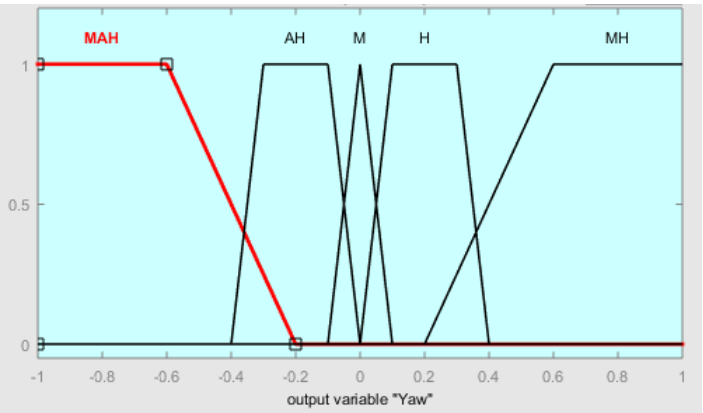
Figura 54: Funciones de pertenencia para derivada de error en Yaw **(deYaw)**.



Fuente: Propia del autor

Para la variable de salida Yaw se le asigna cinco conjuntos borrosos; las etiquetas de esta variable de control son: Sentido Muy Anti horario (SMAH), Sentido Anti horario (SAH), Mantenerse (M), Sentido Horario (SH) y Sentido Muy Horario (SMH). El intervalo de valores de salida es de $[-1, 1]$.

Figura 55: Funciones de pertenencia para derivada de error en Yaw **(deYaw)**.

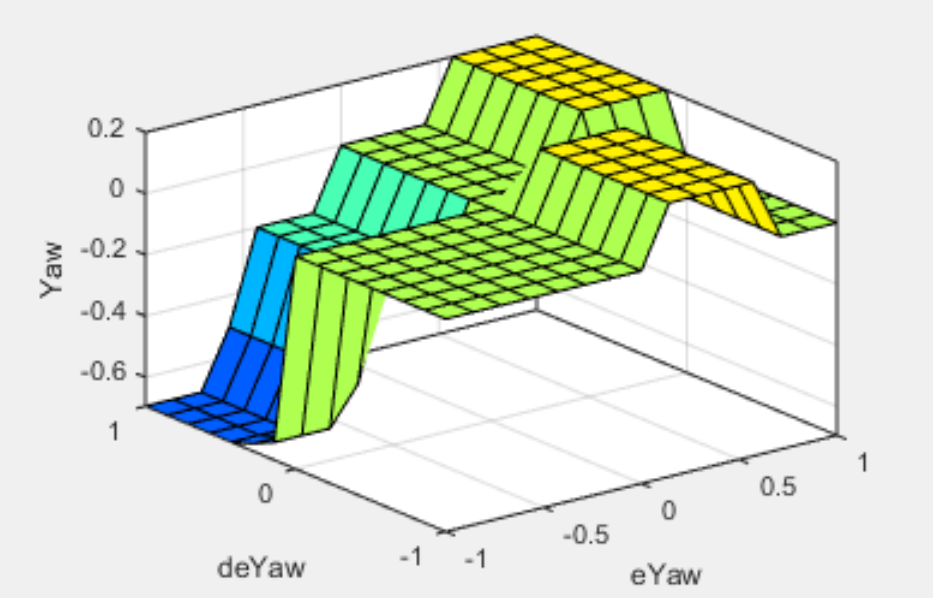


Fuente: Propia del autor

deYaw/eYaw	MH	H	C	AH	MAH
Positivo	SMAH	SAH	SH	SH	SMH
Neutro	SMAH	SAH	M	SH	SMH
Negativo	SMAH	SAH	SAH	SH	SMH

Reglas de inferencia de Yaw.

Figura 56: Gráfico de superficie generada por las composiciones de las reglas.



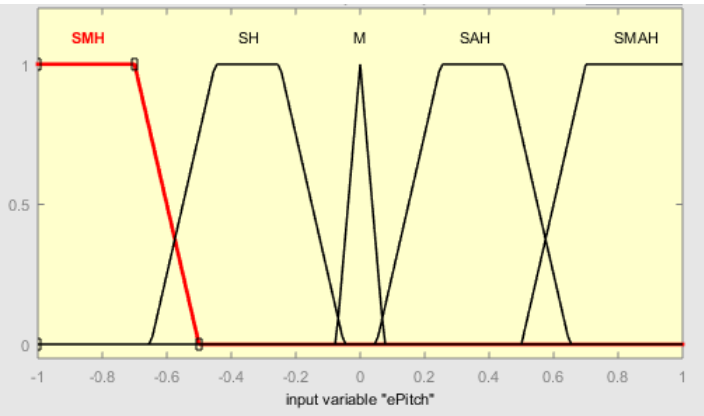
Fuente: Propia del autor

■ **Control de Pitch**

Para el control en Pitch, al igual que en Roll, son necesarias nuevas ecuaciones físicas que modelen el giro y respondan en el desplazamiento por en eje X del quadrotor. Este movimiento también afecta la altura, pero no a Roll ya que las acciones de control actúan en motores diferentes. Las variables de entrada nuevamente son el error en el ángulo Pitch $(ePitch)$ y su respectiva derivada $(dePitch)$.

La primera variable de entrada $ePitch$ consta de cinco conjuntos borrosos, estos han sido llamados; Sentido Muy Anti horario (SMAH), Sentido Anti horario (SAH), Mantenerse (M), Sentido Horario (SH) y Sentido Muy Horario (SMH). En referencia a si el giro en el eje Y que se quiere realizar es en sentido horario, anti horario o si se quiere mantener el rumbo, tal cual como se muestra en la Figura57.

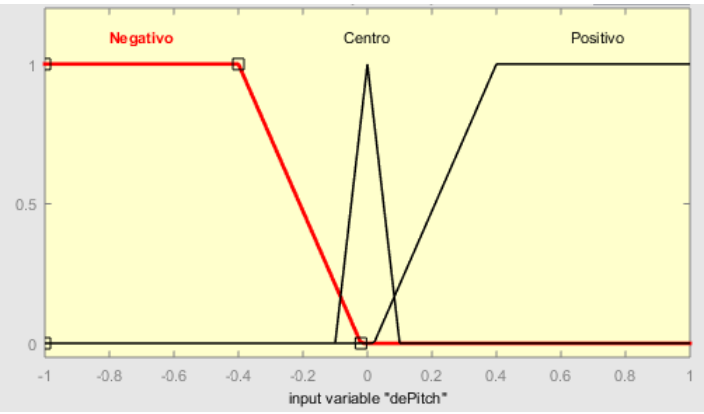
Figura 57: Funciones de pertenencia para error en Pitch **(ePitch)**.



Fuente: Propia del autor

Los tres conjuntos borrosos de la variable $dePitch$ están divididos en un rango de $[-1, 1]$, estos se han llamado Positivo, Centro y Negativo. Etiquetas lingüísticas que recogen si el valor de la derivada del error en Pitch es positiva, negativa o aproximada a cero.

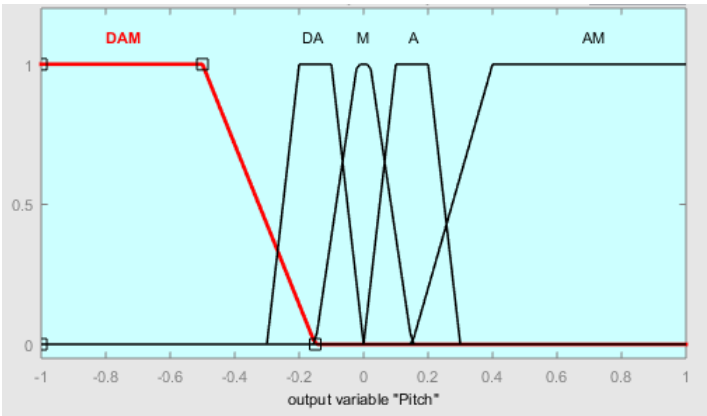
Figura 58: Funciones de pertenencia para la derivada del error en Pitch **(dePitch)**.



Fuente: Propia del autor

La variable de salida $Pitch$ esta cuenta con rango de $[-1, 1]$ repartiéndose así por cinco conjuntos borrosos. Las etiquetas de esta variable de control son: Desacelerar Mucho (DAM), Desacelerar (DA), Mantenerse (M), Acelerar (A) y Acelerar Mucho (AM), como se muestra en la figura 59.

Figura 59: Funciones de pertenencia para salida de control en Pitch **(Pitch)**.

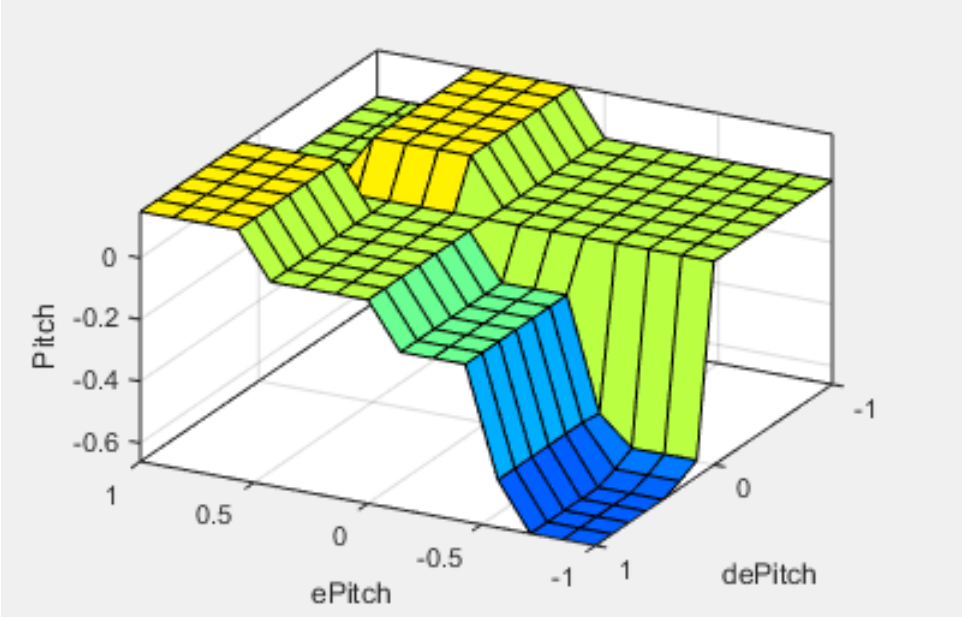


Fuente: Propia del autor

dePitch/ePitch	SMAH	SAH	C	SH	SMH
Positivo	AM	A	A	DA	DAM
Neutro	AM	A	M	DA	DAM
Negativo	AM	A	DA	DA	DAM

Reglas de inferencia de Pitch.

Figura 60: Gráfico de superficie generada por las composiciones de las reglas.



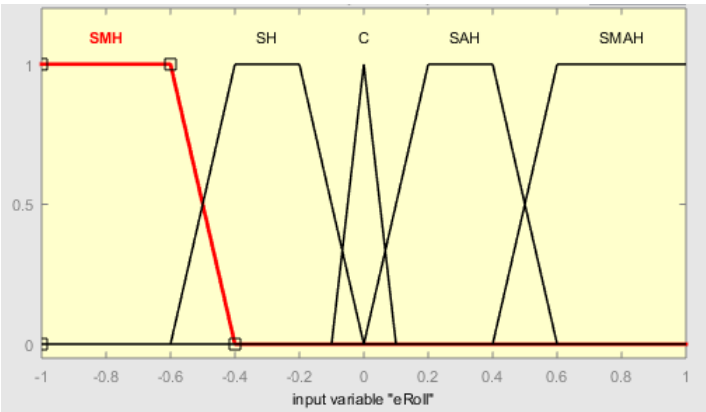
Fuente: Propia del autor

■ **Control de Roll**

Como se mencionó anteriormente el control de Roll necesita de ecuaciones físicas dado que es necesario controlar los movimientos rotacionales y provocar respuestas en el desplazamiento lateral del quadrotor. También es necesario hacer las calibraciones oportunas equilibrando desajustes que pudieran afectar a la altura.

Para la variable de entrada $eRoll$ se ha particionado en cinco conjuntos borrosos. Definido en los extremos con funciones de pertenencia trapezoidales y la central de forma triangular. A cada conjunto borroso se la asigno una etiqueta lingüística: Sentido Muy Horario (SMH), Sentido Horario (SH), Centro (C), Sentido Anti horario (SAH) y Sentido Muy Anti horario. El rango de valores para esta variable lingüística se muestra en la figura61.

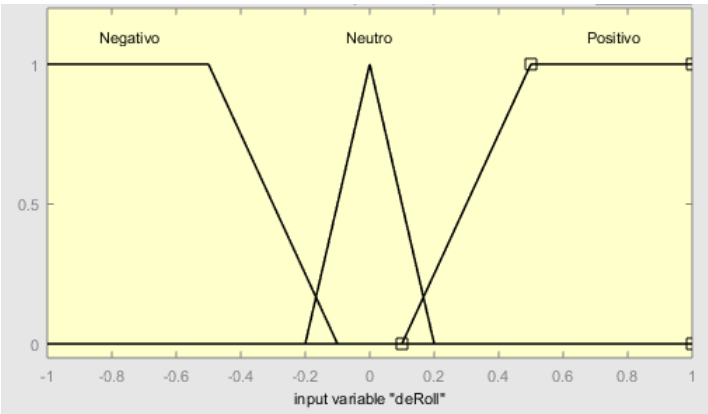
Figura 61: Funciones de pertenencia error en Roll **(eRoll)**.



Fuente: Propia del autor

Los tres conjuntos borrosos de la variable de entrada $deRoll$ tienen las siguientes etiquetas lingüísticas: Negativo, Neutro y Positivo.

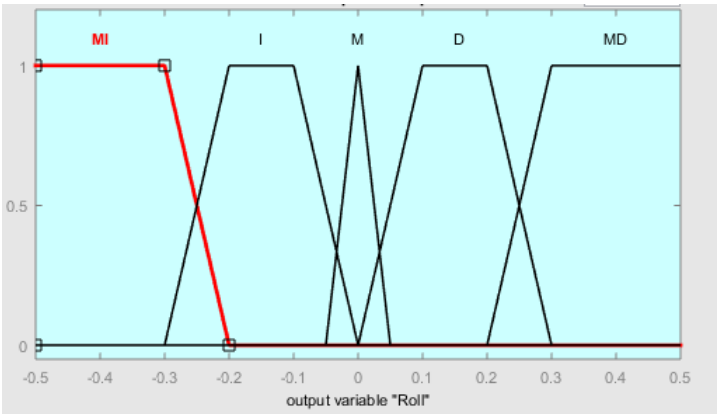
Figura 62: Funciones de pertenencia de derivada error en Roll **(deRoll)**.



Fuente: Propia del autor

La variable de salida Roll cuenta de cinco conjuntos borrosos, las etiquetas de esta variable de control son: Muy Izquierda, Izquierda (I), Mantenerse (M), Derecha (D) y Muy Derecha (MD). En la figura 63 se muestra el rango de valores y sus particiones con las cinco funciones de pertenencia. El dominio de salida es menor pues represente incrementos de potencia a añadir.

Figura 63: Funciones de pertenencia para salida de control en Roll **(Roll)**.

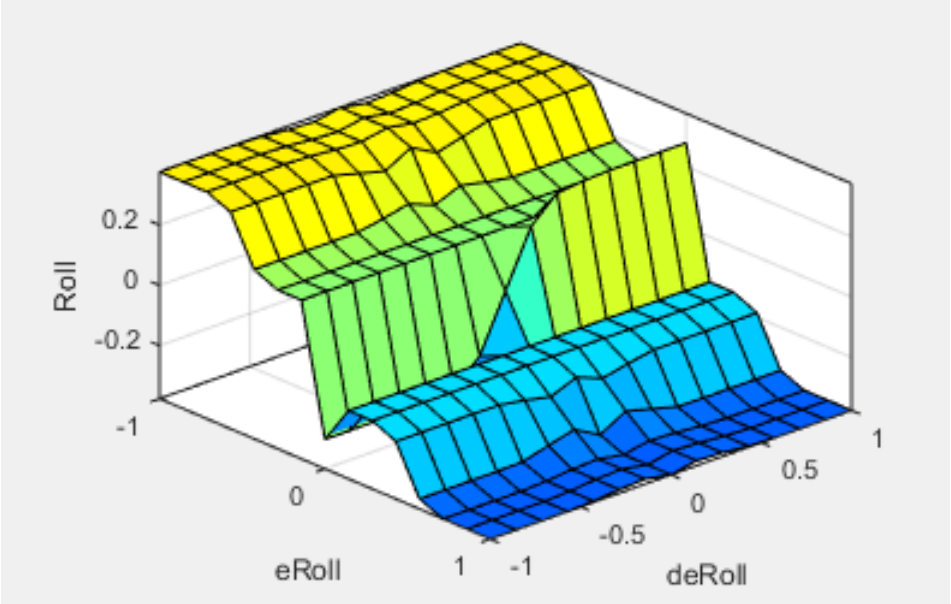


Fuente: Propia del autor

deRoll/eRoll	SMH	SH	C	SAH	SMAH
Positivo	MD	D	D	I	MI
Neutro	MD	D	M	I	MI
Negativo	MD	D	I	I	MI

Reglas de inferencia de Roll.

Figura 64: Gráfico de superficie generada por las composiciones de las reglas.



Fuente: Propia del autor

4.1.2. Controladotes PID

Como se mencionó con anterioridad, los métodos utilizados para diseñar los controladores PID son Ziegler Nichols y Lugar geométrico de las raíces. Para el diseño de los controladores por Ziegler Nichols se desarrolló un algoritmo el cual ingresando la planta y localizando el punto de corte de las raices se determina el valor de sus ganancias. El diseño por lugar geométrico de las raíces esta dado por la función **rltool** de Matlab la cual por medio del posicionamiento de los polos determinamos las ganancias de nuestro controlador.

■ Método de Lugar de las raíces

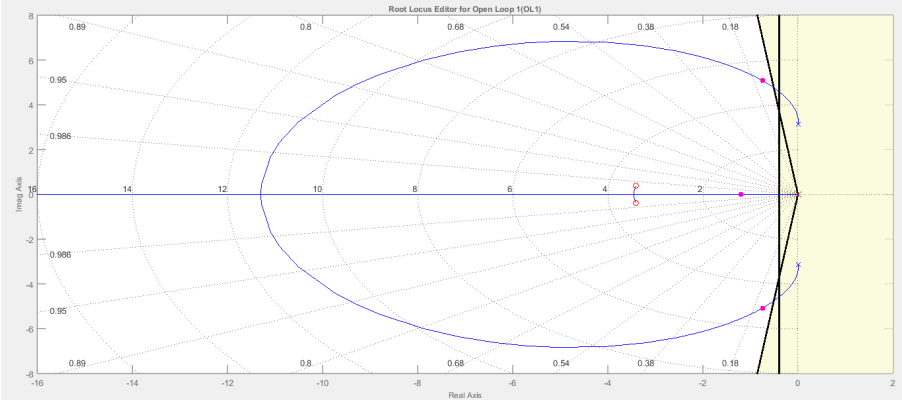
El desarrollo de este método se determina mediante la herramienta **rltool** de Matlab, tomándose como entrada una señal de paso en cada uno de los controladores. Partiendo de cada una de las funciones de transferencia obtenidas por nuestro vector de estados, se diseña cada controlador donde se toma como referencia los valores iniciales del modelo sin ningún tipo de variación en los valores a controlar.

– Control PID Alruta

La asignación de polos para este controlador está definida por los siguientes parámetros:

- Tiempo pico de $1s$
- Sobre impulso de 25 %.
- Tiempo se asentamiento de $10s$.
- Error de estado estable de 0.01 %.

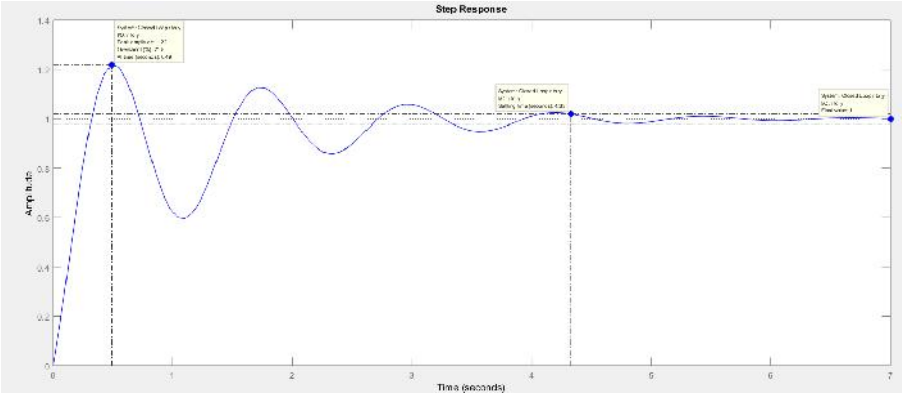
Figura 65: Root Locus Editor Altura.



Fuente: Propia del autor

El controlador obtenido para la señal de altura esta dado por las ganancias $K_p = 18{,}3$, $K_i = 31{,}6$ y $K_d = 2{,}69$ dando como resultado la repuesta de la figura 66.

Figura 66: Respuesta a controlador LGR Altura.



Fuente: Propia del autor

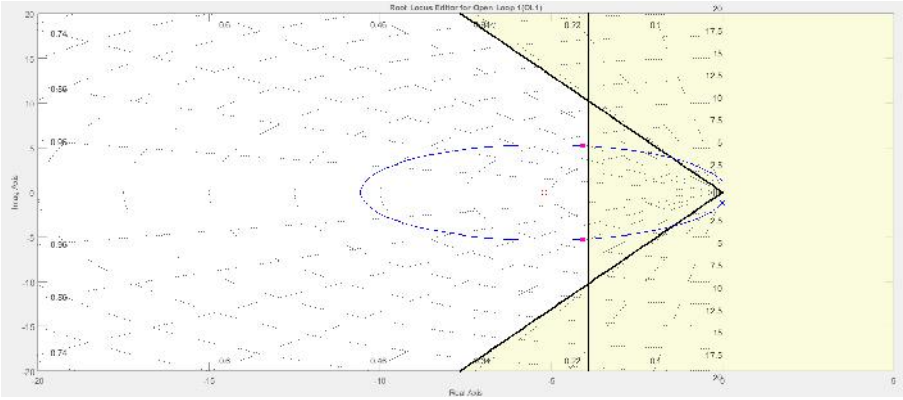
Como se muestra en la figura 66 la cual tiene como entrada un $step$ el tiempo pico es de $1{,}22s$, el sobre impulso de $21{,}8\,\%$ y un tiempo de asentamiento de $4{,}33s$.

– **Control PID Yaw**

La asignación de polos para este controlador está definida por los siguientes parámetros:

- Tiempo pico de $0{,}3s$
- Sobre impulso de 25 %.
- Tiempo se asentamiento de $1s$.
- Error de estado estable de 0.01 %.

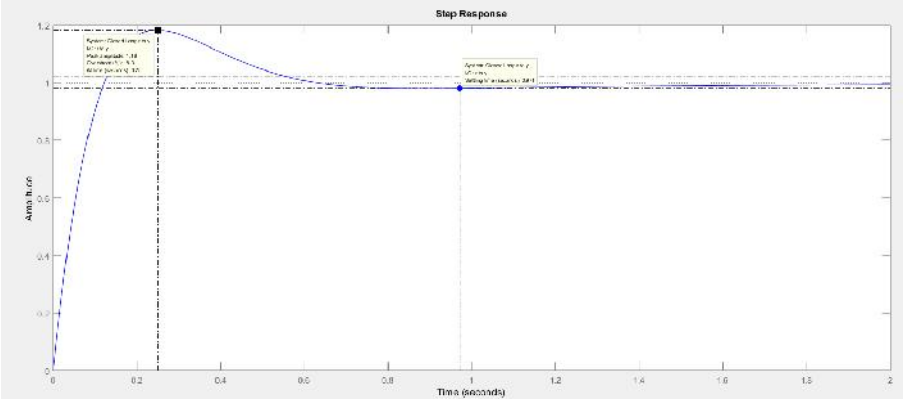
Figura 67: Root Locus Editor Yaw.



Fuente: Propia del autor

El controlador obtenido para la señal de altura esta dado por las ganancias $K_p = 2{,}3$, $K_i = 0{,}04$ y $K_d = 0{,}12$ dando como resultado la repuesta de la figura 68.

Figura 68: Respuesta a controlador LGR Yaw.



Fuente: Propia del autor

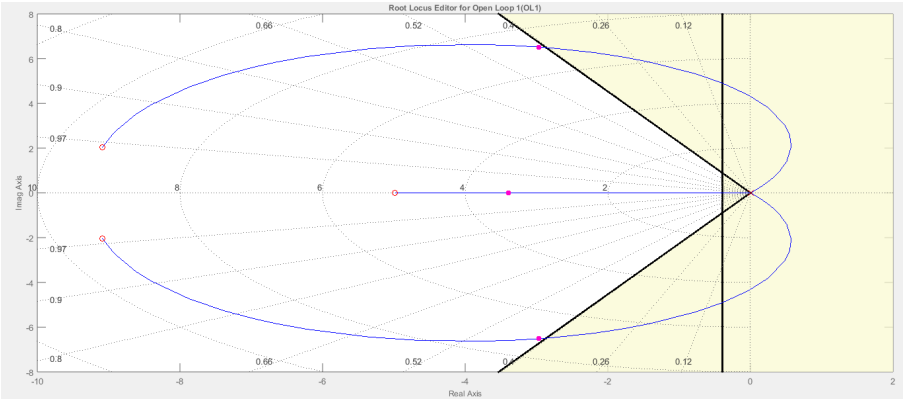
Como se muestra en la figura 68 la cual tiene como entrada un $step$ el tiempo pico es de $0{,}25s$, el sobre impulso de $18{,}3\,\%$ y un tiempo de asentamiento de $0{,}971s$.

– **Control PID Pitch**

La asignación de polos para este controlador está definida por los siguientes parámetros:

- Tiempo pico de $0{,}5s$
- Sobre impulso de 20 %.
- Tiempo se asentamiento de $1{,}5s$.
- Error de estado estable de 0.01 %.

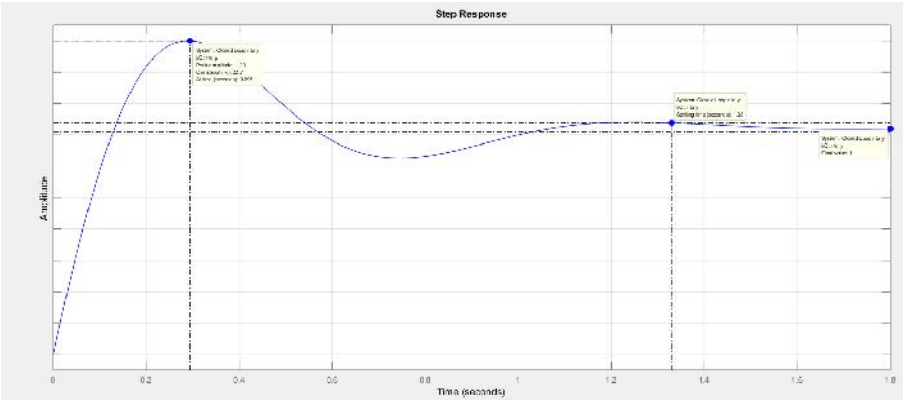
Figura 69: Root Locus Editor Pitch.



Fuente: Propia del autor

El controlador obtenido para la señal de altura esta dado por las ganancias $K_p = 3{,}12$, $K_i = 0{,}29$ y $K_d = 4{,}65$ dando como resultado la repuesta de la figura 70.

Figura 70: Respuesta a controlador LGR Pitch.



Fuente: Propia del autor

Como se muestra en la figura 66 la cual tiene como entrada un $step$ el tiempo pico es de $0{,}29s$, el sobre impulso de $22{,}7\,\%$ y un tiempo de asentamiento de $1{,}33s$.

4.2. Algoritmo de procesamiento de imágenes en serie

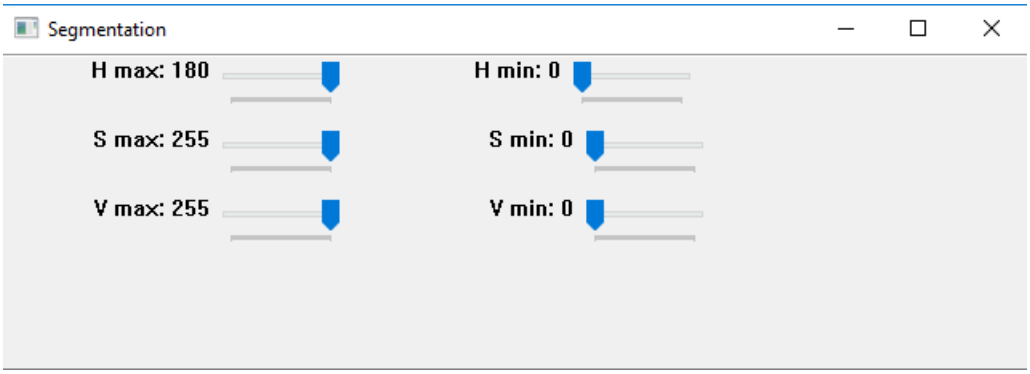
El desarrollo del algoritmo en serie se llevó acabo con las librerías OpenCV para la detección del objeto por color. Como se mencionó anteriormente, la cámara frontal del AR Drone 2.0 se puede configurar con dos resoluciones. Para este algoritmo se optó por la resolución de 640x360 de la imagen de entrada. El software utilizado para la captura de imagen es cvdrone que ya tiene incluido la librería OpenCV.

Para comenzar las imágenes transmitidas por el AR Drone son almacenadas en una clase *Mat*. Con esta clase no es necesario asignar o manipular memoria y contiene dos tipos de datos: El header de la matriz que contiene información como el tamaño de la matriz y el apuntador a la matriz que contiene los valores de los pixeles. Luego, se crea otra clase Mat con nombre HSV y se utiliza la función *cv::cvtColor* para convertir la imagen de BGR a HSV. Esta función tiene argumentos que son la imagen de entrada, la imagen de salida y el código de espacio de color *BGR2HSV*.

Como el algoritmo es a base de color, el usuario tiene la opción de cambiar el color segmentado mediante trackbars. Con estas trackbars el usuario puede controlar los valores H, S y V de la imagen. Para llevar a cabo lo anterior, se utilizan las funciones *Scalar* y *inRange* con motivo de crear un límite inferior y un límite superior para los valores HSV. Estas tres variables son de 8 bits ósea que tienen valores de 0 a 255

pero para la variable H se tiene un valor de 0 a 180. En la figura 71 se puede ver las trackbars.

Figura 71: Trackbars de segmentación



Fuente: Propia del autor

El resultado de la segmentación es una imagen binaria donde los pixeles blancos corresponden a el color deseado o al número 1 y los pixeles negros a otros colores o el número 0. Pasa muchas veces que la imagen binaria contiene ruido y tiende a afectar los resultados. Una solución para este problema es aplicar operaciones morfológicas como la erosión y la dilatación. Primero se aplica una erosión con un elemento estructurante cuadrado de 3x3 para eliminar pixeles blancos alrededor del objeto. Este elemento estructurante como se muestra en la figura 72 pasa por cada pixel en la imagen hace una operación lógica AND, es decir, si todos los pixeles en la vecindad de la imagen son blancos el resultado va a ser un pixel blanco como resultado. Al contrario, si existe solo un pixel negro en la vecindad el resultado es un pixel negro. Esta operación morfológica puede ser aplica varias veces.

Figura 72: Elemento estructurante

1	1	1
1	1	1
1	1	1

Fuente: Propia del autor

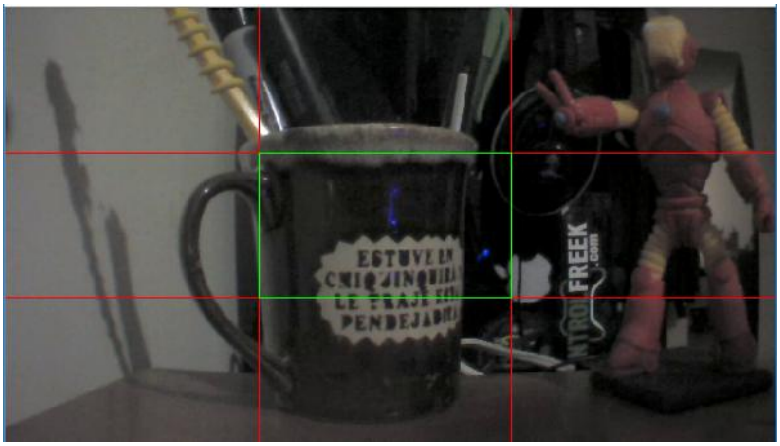
Luego de aplicar la operación morfológica erosión, se aplica la operación morfológica dilatación. La razón para aplicar esta operación es de remover el ruido que se encuentra adentro del objeto segmentado. Esta operación morfológica es similar a la operación lógico OR, donde sí se encuentra un pixel blanco en la vecindad el resultado es un pixel blanco y si todos los pixeles son negros el resultado es un pixel negro. Para esta operación se utiliza el mismo elemento estructurante que se usó en la erosión y se puede aplicar varias veces.

Después de tener la imagen con menos ruido se aplica la función *cv::findContours()*. La función tiene cinco argumentos los cuales son: la imagen binaria, vector de puntos, jerarquía, modo de adquisición de contornos y método de aproximación. En el algoritmo no se tiene en cuenta la jerarquía, para modo de adquisición de utiliza *CV RETR LIST* que adquiero los contornos y los coloca en una lista. Adicional a la función, se crea una variable *contourindex* el cual obtiene el numero de contornos que fueron encontrados.

Ya teniendo lo contornos, se calculan los momentos para el centroide del objeto. La función de OpenCv para ello es *cv::moments()* que tiene dos argumentos que son los puntos del contorno en el cual es de interes y el segundo un tipo de dato **bool** que es true si la imagen es binaria y false si la imagen es de otro tipo. Al momento de ejecutar esta función automaticamente calcula los momentos de hasta tercer orden, pero para el centroide solo se necesita las de primero orden. Para acceder a esta informacion la estructura de la función *moments* se accede por *moments.m00, moments.m10 y moments.mo1* donde, $m00$ es la longitud o área del objeto, $m10$ es los momentos en x y $m01$ los momentos en y. Se crean dos variables $markerx$ y $markery$ donde se divide los momentos en x y y en el área lo que da como resultado las coordenadas del centroide del objeto en la imagen.

Por ultimo, se imprime las coordenadas del centroide del objeto en la imagen y se divide la imagen en secciones para el controlador. La división de la imagen se puede ver en la figura 73.

Figura 73: División Imagen



Fuente: Propia del autor

4.3. Algoritmos de procesamiento de imágenes en paralelo

En la siguiente sección se explicará el desarrollo de la conversión de una imagen de RGB a escala de grises, el filtro gaussiano y filtro sobel en paralelo. Se presenta diagramas de flujo para cada algoritmo tanto para la CPU y la GPU. Enseguida se explica brevemente términos de la arquitectura que son utilizados en los algoritmos y las variables creadas:

- **uchar4** - tipo de dato de CUDA el cual almacena 4 datos tipo unsigned chars. El tamaño es de 4 bytes y se accede cada dato por *.x, .y, .z, .w*.
- **threadIdx** - identificador para cada thread en un bloque
- **blockIdx** - identificador del bloque en un grid
- **blockDim** - numero de threads en un bloque
- **HOST** - memoria de la CPU
- **DEVICE** - memoria de la GPU
- **kernel** - función que se ejecuta en la GPU
- **numCols** - numero de columnas de la imagen de entrada
- **numRows** - numero de filas de la imagen de entrada

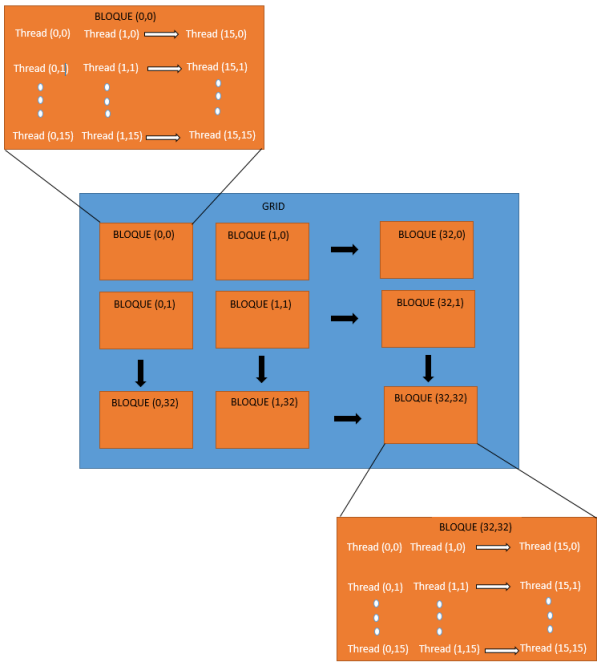
Para los tres algoritmos se utiliza la misma configuración de bloques y threads. Así mismo, cada bloque es de tamaño 16x16, es decir, 256 threads por bloque y el número de bloques es determinado por las siguientes ecuaciones para bloques en dos dimensiones:

$$x = numCols/16 + 1$$

$$y = numRows/16 + 1$$

Puede pasar que se configuren más threads que pixeles en una imagen, pero no tendrá ningún efecto en la ejecución del algoritmo. La figura 74 ilustra la configuración de bloques y threads para una imagen de tamaño 512x512.

Figura 74: Configuración bloques y threads una imagen de 512x512



Fuente: Propia del autor

4.3.1. Conversión RGB a escala de grises

La conversión de una imagen de color RGB a una imagen a escala de grises se puede interpretar como el promedio de tres canales de una imagen o el resultado de intensidad de luz al lugar de color. Un método simple para la conversión esta dada por la siguiente ecuación:

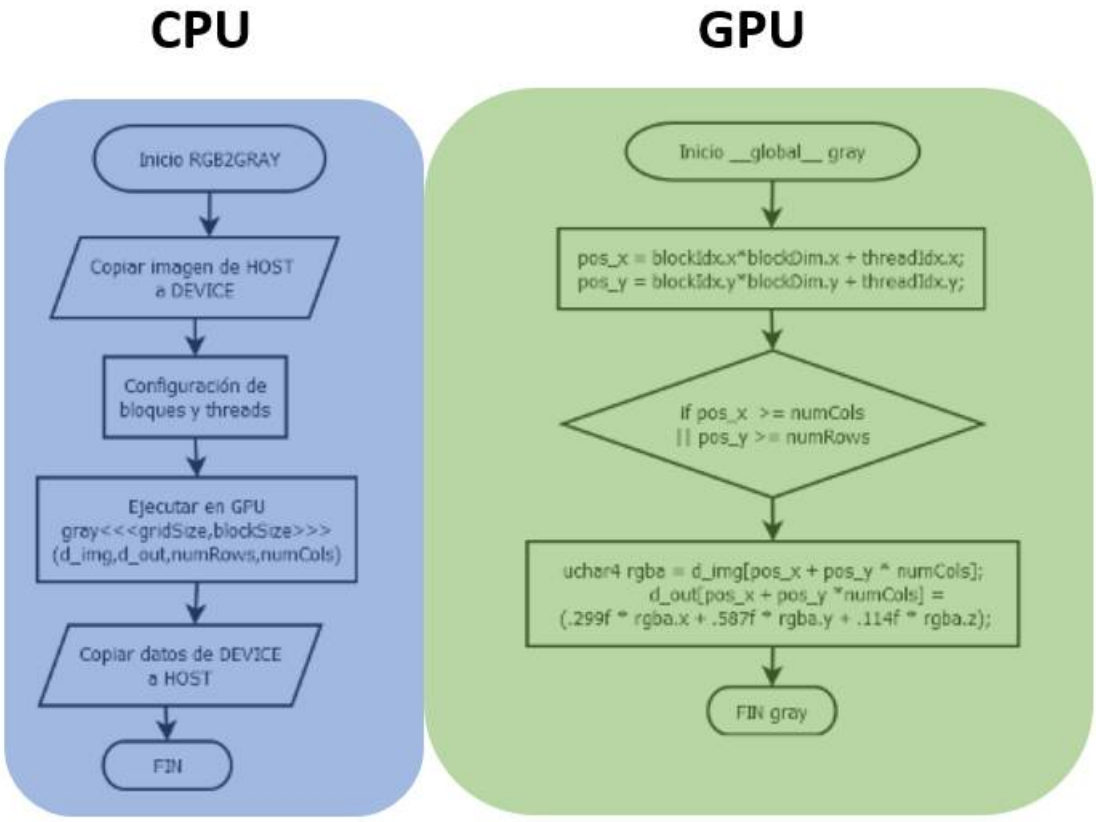
$$GRAY(x, y) = \frac{R(x, y) + G(x, y) + B(x, y)}{3}$$

donde, x y y es la posición del pixel y RGB los 3 canales. Otra forma más sofisticada es recomendada por la NTSC(National Television System Committee)[55] ya que el ojo responde con mayor intensidad al verde seguido por el rojo y luego el azul. Con la siguiente ecuación se desarrolla el algoritmo:

$$gray = 0{,}299 * r + 0{,}587 * g + 0{,}114 * b$$

En la figura 75 se presenta el diagrama de flujo de la ejecución del algoritmo. Para comenzar la librería OpenCV se utiliza para adquisición de la imagen y sus datos. Al adquirir esta imagen la se convierte a RGBA para acceder a los pixeles con el tipo de dato **uchar4**, es decir, con *.x* se accede al canal R, *.y* se accede al canal G y *.z* se accede al canal B. En este algoritmo no se tiene en cuanta el canal *ALPHA*.

Figura 75: Diagrama de flujo para conversión RGB a escala de grises



Fuente: Propia del autor

Como se puede observar en el diagrama de flujo se copia la imagen de la memoria de la CPU a la GPU con la funcion *cudaMemcpy*, esto paso se hace después

de asignar memoria de la GPU. Después se configura los bloques y threads como se explico al comienzo del capitulo y se ejecuta el kernel gray. Los argumentos de este kernel son *d_img* que es la imagen de entrada, *d_out* es la imagen de salida, *numRows* es el numero de filas y *numCols* es el numero de columnas de la imagen.

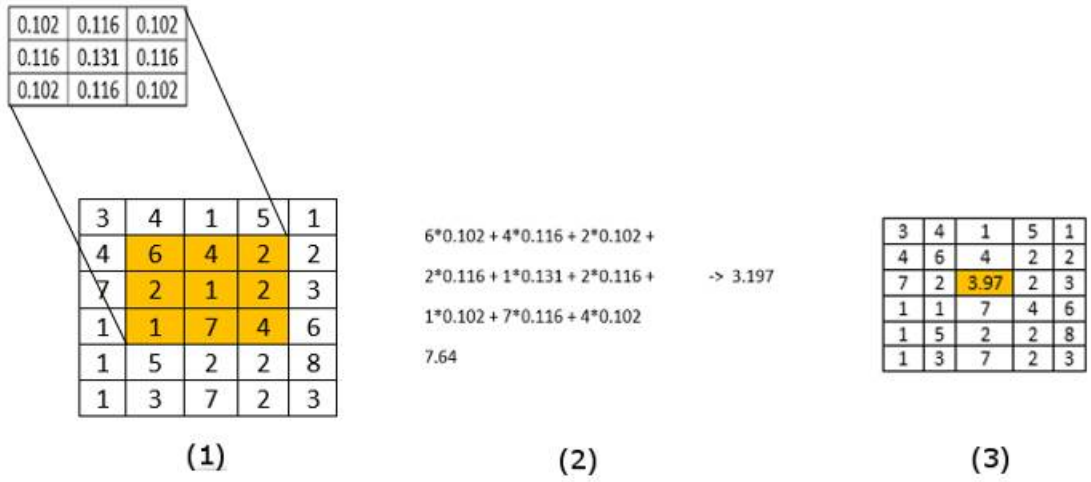
Ahora se va explicar la configuración de la función *global* que se ejecuta en la GPU. Primero, se crean dos variables tipo *int* **pos_x** y **pos_y** para la posición absoluta de la imagen. Luego declaramos un **if statement** para tener el mismo número de filas y columnas que la imagen de entrada. Después, se crea una variable *rgba* de tipo **uchar4** para leer la imagen de entrada y almacenar el valor de cada pixel en cada canal. Por último, se accede a la variable *rgba* y sus componentes para calcular el resultado y pasarlo a la imagen de salida. Se debe agregar que la operación $[pos\_x + pos\_y * numCols]$ se puede interpretar como la posición global de cada pixel.

Finalizando, la imagen de salida se copia de vuelta a la memoria de la CPU para visualizar el resultado.

4.3.2. Filtro Gaussiano

La idea principal del filtro gaussiano es remover el ruido de la imagen y eliminar altas frecuencias. Los parámetros de este filtro son la desviación estándar σ y el tamaño del filtro, estos dos paramentos pueden ser modificados por el usuario. Este filtro es un tipo de convolución, en procesamiento de imágenes esta convolución es el proceso de agregar cada elemento de la imagen a sus vecinos locales, ponderado por el filtro. En la imagen 76 se muestra esta operación donde se tiene el filtro y la imagen; el resultado se ve en la nueva imagen donde solo se hizo para un pixel.

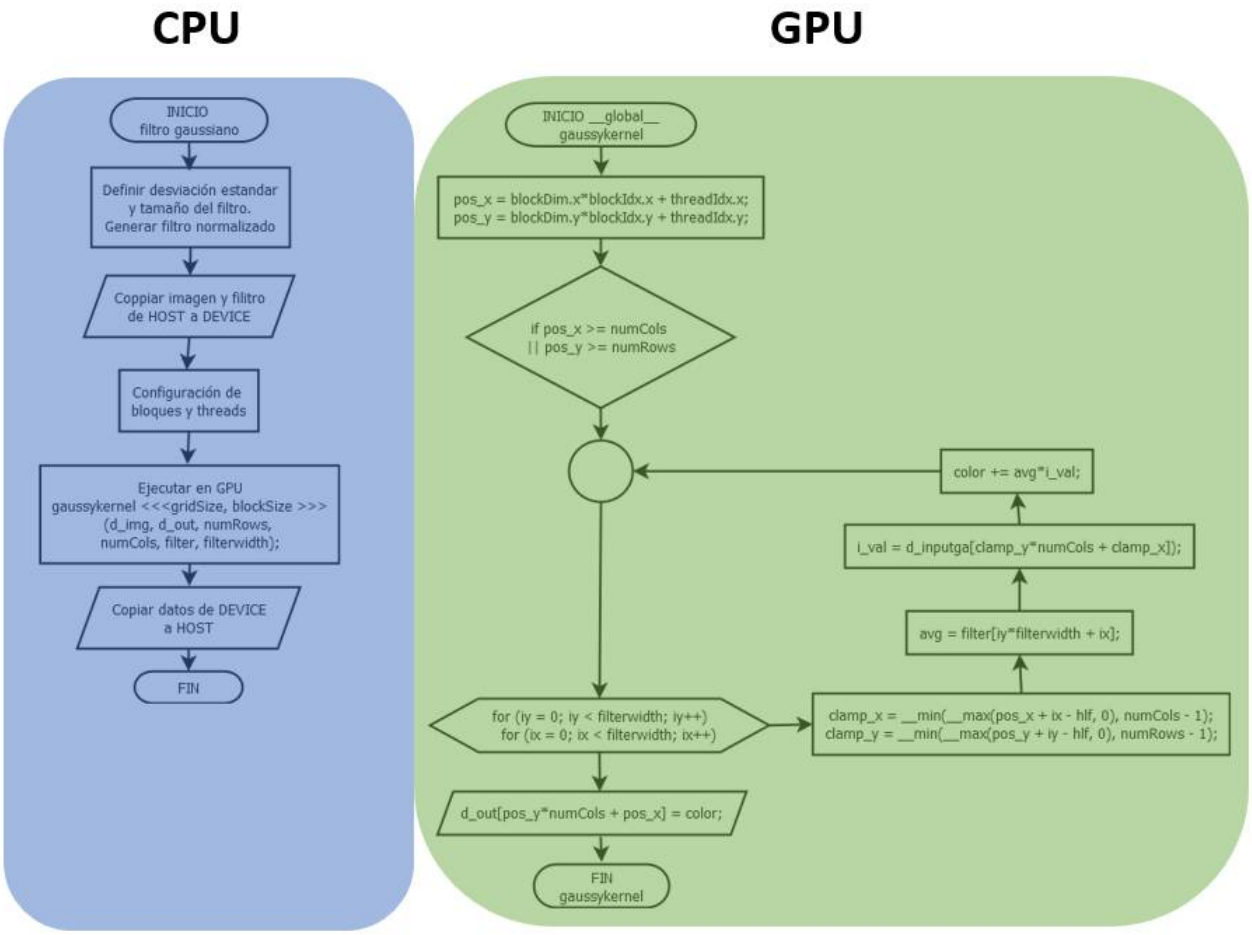
Figura 76: Convolución



Fuente: Propia del autor

La figura 77 se presenta el diagrama de flujo para el filtro gaussiano. El primer paso es generar el filtro en la CPU y almacenarlo en un *array* de tipo *float*. El filtro se puede interpretar como una imagen de tamaño NxN, donde se recomienda que N sea un número impar; para el origen del filtro se elige el centro, es decir la posición (0,0).

Figura 77: Diagrama de flujo para el filtro gaussiano

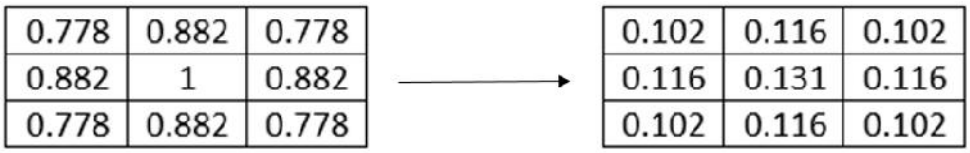


Fuente: Propia del autor

Los valores del filtro son dados por la ecuación en la sección 3.2.11 donde *x* y *y* son las posiciones del filtro y σ la desviación estándar. Ya teniendo los valores del filtro se normaliza para evitar que la imagen de salida se vea pixeleada; para normalizar se divide 1 entre la sumatoria de todos los valores del filtro; ya con el valor obtenido se multiplica por cada elemento del filtro. Gracias a la normalización, se obtiene una mejor implementación del filtro. Como se observa en la figura 78, se

nota la diferencia del filtro normalizado y se puede notar que el pixel del centro tiene más peso que los pixeles vecinos.

Figura 78: Normalización de filtro



Fuente: Propia del autor

Luego de tener el filtro se asigna memoria y se copia los datos de la imagen de entrada y el filtro. La configuración de threads y bloques es la igual a la que se utilizó para la conversión de RGB a escalas de grises. Ahora se ejecuta el kernel *gaussykernel*, el argumento *filter* es el filtro creado en la CPU y *filterwidth* el tamaño de dimensión del filtro. Se crean las mismas variables *pos_x* y *pos_y* para tener la posición absoluta de la imagen. Se crea un *for loop* para adquirir el valor del filtro con la variable *avg*. Para el problema de cuando el filtro se sale del límite de la imagen cuando opera en los bordes se crean las variables *clamp_x* y *clamp_y*. Estas variables reemplazan el valor del pixel que está al lado ya sea de la fila o la columna. Así mismo, la variable *i_val* adquiere la posición del pixel global y con la variable *color* se almacena la multiplicación de *avg* por *i_val*.

Para comprender mejor, el filtro está almacenado en un *array* de N posiciones, con el for loop se adquiere el valor del filtro comenzando en la posición [0] y se multiplica por cada thread que contiene el valor del pixel. El resultado se almacena en una variable y se repite hasta llegar a la última posición $N-1$ del *array*.

Finalmente, se escribe el resultado en la imagen de salida d_out y se copia el resultado de vuelta a la memoria de la CPU. Este diagrama de flujo se aplica a una imagen a escala de grises, para aplicar el algoritmo a una imagen RGB se tiene que separar los canales, aplicar el filtro a cada canal y por ultimo combinar los tres colores. También, el algoritmo se puede adaptar a la aplicación requerida ya que si se requiere un suavizado más alto se puede incrementar la desviación estándar σ.

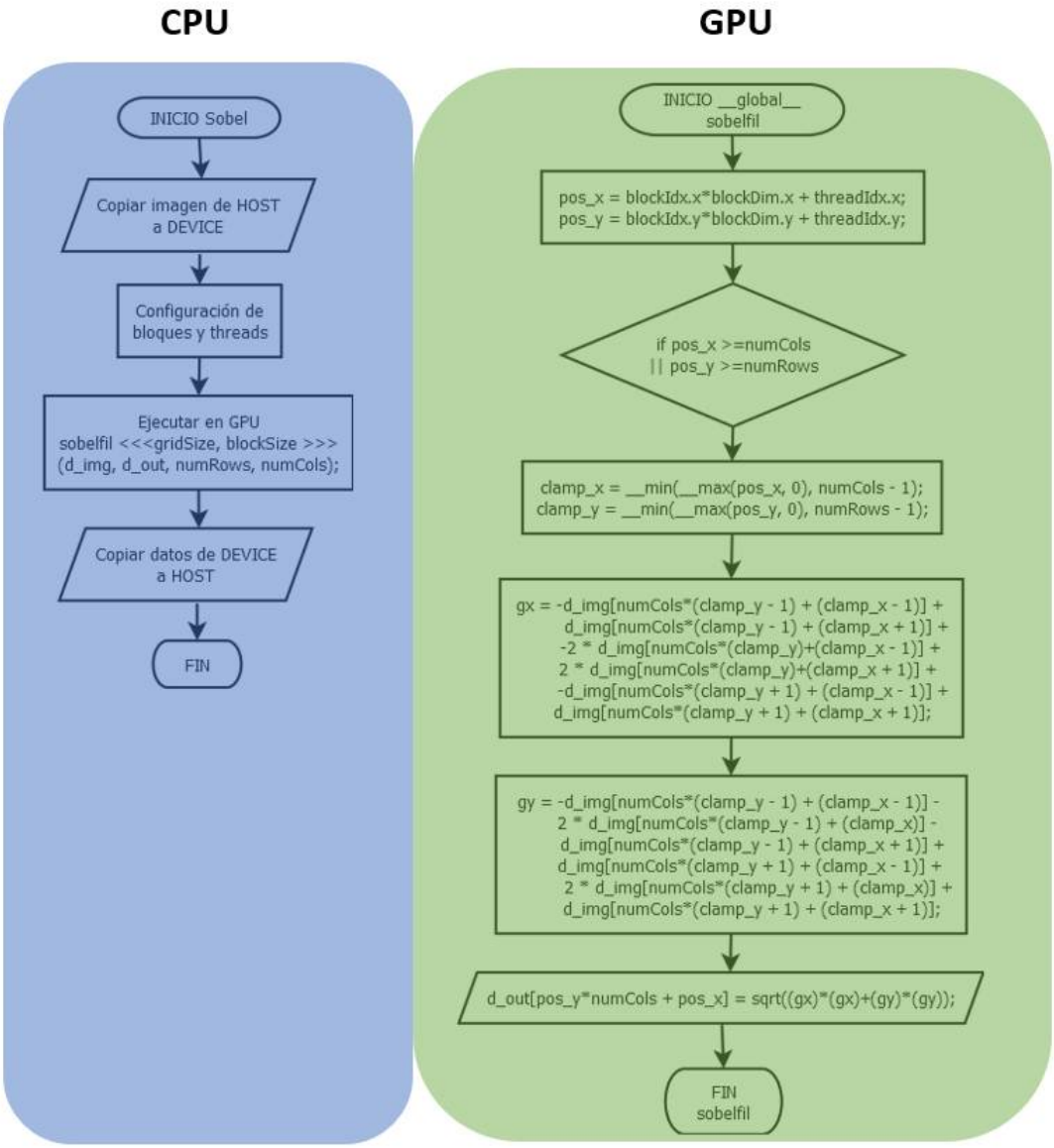
4.3.3. Filtro Gradiente Sobel

La aplicación del filtro sobel es encontrar los bordes en una imagen en escala de grises. Como este filtro es sensible al ruido se va a aplicar a la imagen preprocesada por el filtro gaussiano que se explicó en la sección anterior.

El diagrama de flujo del filtro sobel se muestra en la figura 79 y sigue la misma

idea del filtro gaussiano calculando la respuesta del pixel central. Como se aprecia en el diagrama se copia los datos de la CPU a la GPU, se configura los threads y bloques y se ejecuta el kernel *sobelfil*. Los cuatro argumentos del kernel son la imagen de entrada, imagen de salida y el número de filas y columnas de la imagen.

Figura 79: Diagrama de flujo para el filtro sobel



Fuente: Propia del autor

La ejecución en la GPU se crean también las variables *pos_x*, *pos_y* para la posición absoluta de la imagen y *clamp_x*, *clamp_y* para remplazar el pixel si se encuentra fuera del límite de la imagen. Las variables *gx* y *gy* son las derivadas en las

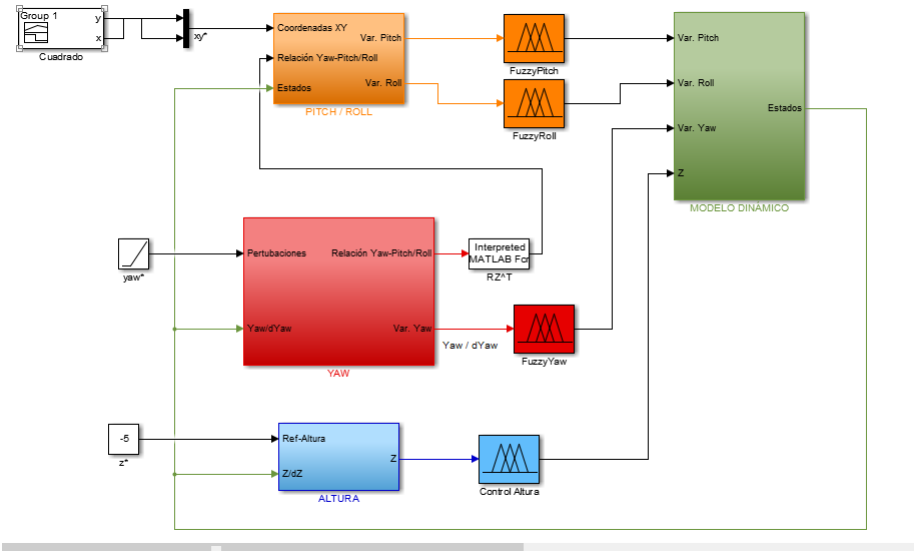
dos dimensiones y simulan los kernels que se explicaron en la sección 3.2.12. Por último, se calcula la magnitud de las dos derivadas y se escribe el resultado en la imagen de salida *d_out*.

Para entender mejor este filtro, es el mismo procedimiento que se ve en la figura 76. Si el filtro está en una zona donde se encuentran pixeles de la misma intensidad su resultado va a ser cercano a cero dejando un pixel oscuro. Al contrario, si se encuentra un cambio de intensidad el resultado va a ser mucho mayor cero y deja un pixel luminoso con valor menor a 255.

5. Análisis de Resultados

Para realizar la validación del modelo se utilizó el Toolbox desarrollado por [56], al cual se le implementaron las ecuaciones descritas en la sección 3.1.2 las cuales contienen la cinemática y dinámica del quadrotor. Para obtener las siguientes respuestas del sistema se implementaron dos tipos de controladores clásicos, el primero de tipo difuso y el segundo PID; el objetivo de estos controladores es que el vehículo sea capaz de seguir un par de trayectorias previamente establecidas, permitiendo realizar comparaciones de respuestas con respecto al tipo de control.

Figura 80: Diagrama de bloques del modelo



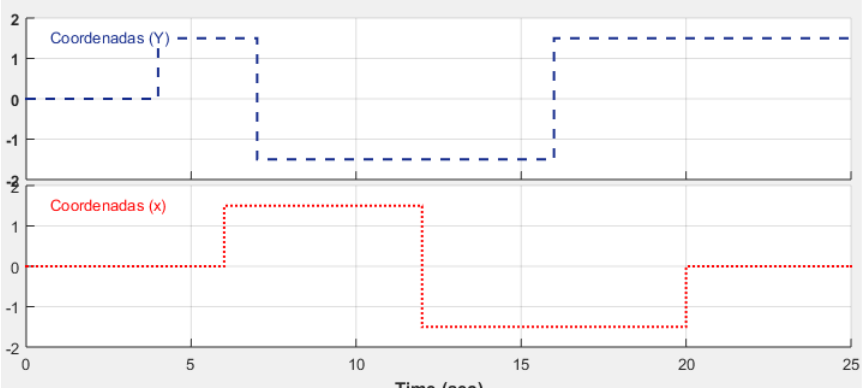
Fuente: Propia del autor

5.1. Primer Prueba (Trayectoria Cuadrada)

En esta prueba se analizarán los ángulos Euler y la altura de manera conjunta. En lo relativo a la altura se requiere que el vehículo se estabilice a 5 metros. Las variables Pitch, Roll y Yaw cambiaran su valor según las coordenadas mostradas en la figura 81, las cuales son realizadas por generadores de señales. Estos giros permitirán el avance, frenado y marcha atrás del quadrotor.

En primera instancia el vehículo se encuentra en el suelo con todos los ángulos en cero con coordenada [0,0,0], al iniciar se dará 3 segundos para que la aeronave despegue e intente llegar a su altura requerida, luego de esto los valores de los ángulos irán cambiando.

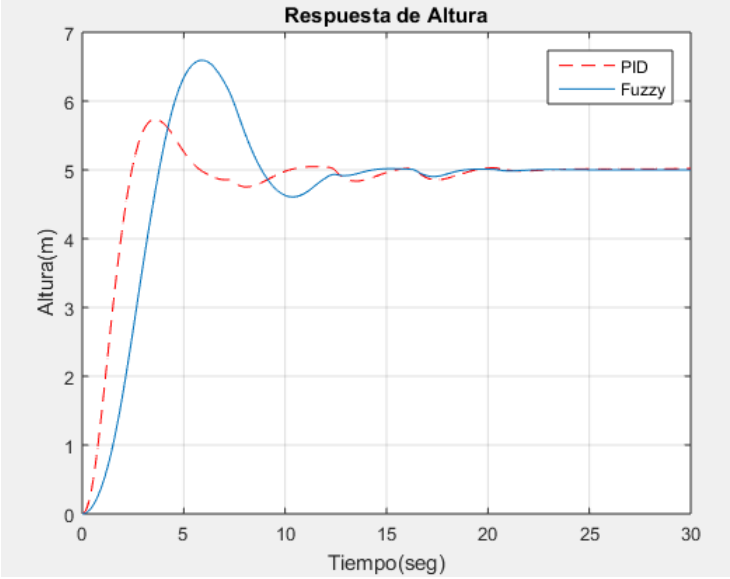
Figura 81: Generador de señales (Trayectoria Cuadrada).



Fuente: Propia del autor

Respuestas para controladores de Altura

Figura 82: Comportamiento de altura.

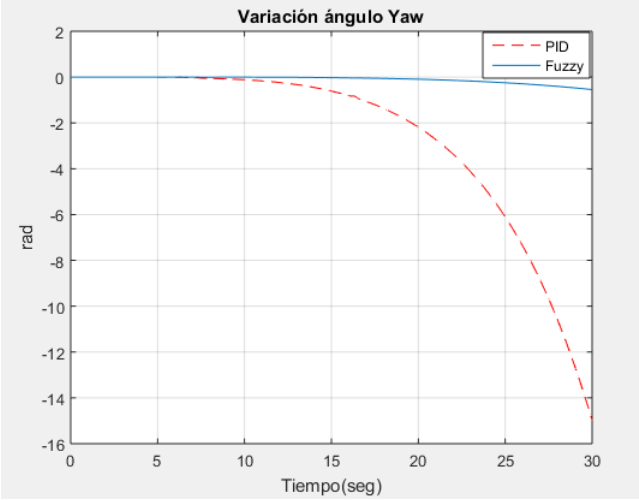


Fuente: Propia del autor

Como se evidencia en la figura 82 la señal de altura del controlador PID tiene un tiempo de establecimiento de aproximadamente 20, casi similar a la obtenida por el control difuso. Ambos se muestran como señales de un sistema de segundo orden sub amortiguadas, con un sobre impulso de 15 % para el control PID y 30 % para el difuso.

Controlador de Yaw

Figura 83: Comportamiento de Yaw

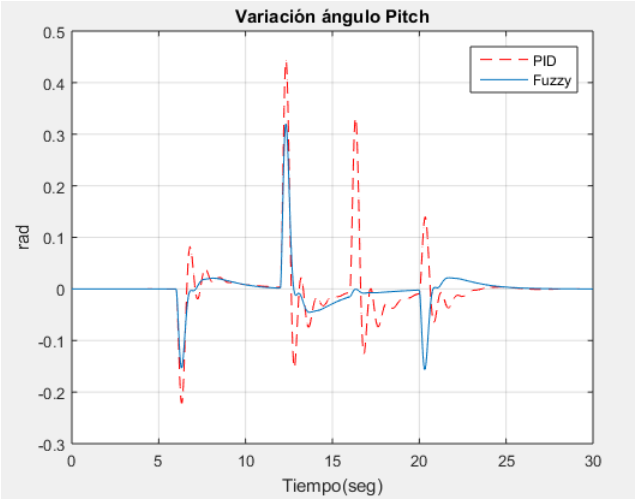


Fuente: Propia del autor

El control de esta variable está diseñado para ajustarse a las variaciones ejercidas por la trayectoria, pero al mismo tiempo para que los ejes del quadrotor no obtengan ninguna desalineación. El comportamiento como se observa en la figura 83 se puede ver que tiende a girar en sentido horario exponencialmente para ambos controladores, evidenciándose un mayor giro en el controlador PID.

Controlador Pitch

Figura 84: Comportamiento de Pitch

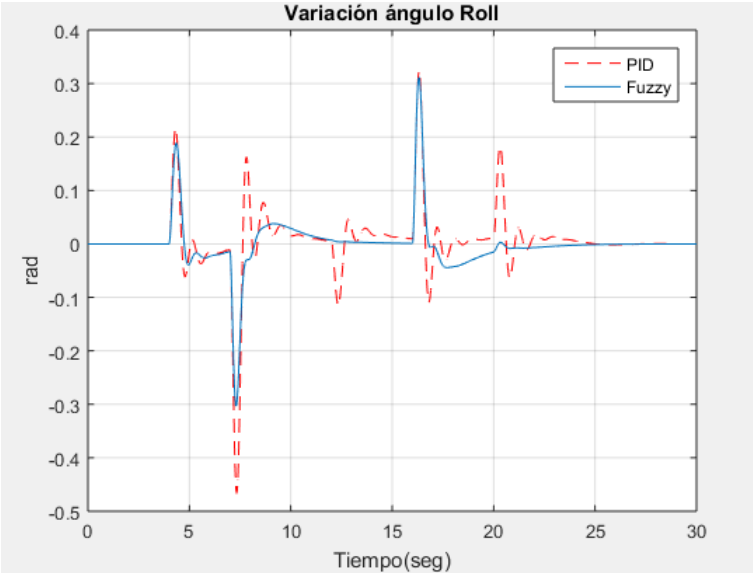


Fuente: Propia del autor

Como se muestra en la figura 84 la variable Pitch no tiene un comportamiento uniforme debido a las variaciones de giro necesarias para que el quadrotor cumpliese con el seguimiento de la trayectoria, dejando ver que el controlador difuso tiende a atenuar mejor los cambios en la señal a comparación del controlador PID.

Controlador Roll

Figura 85: Comportamiento de Roll



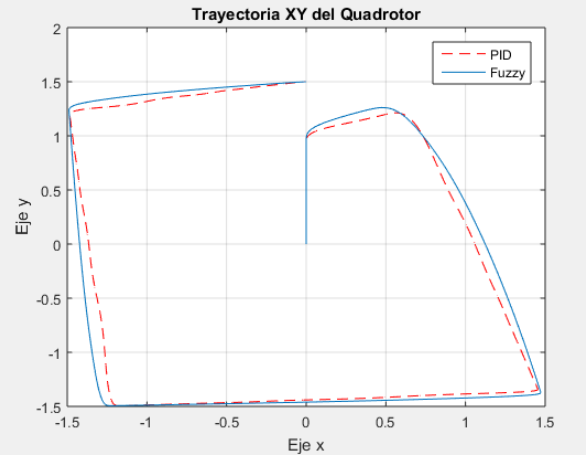
Fuente: Propia del autor

El comportamiento de Roll es similar al de Pitch el cual tiene picos en la señal producidos por la trayectoria, pero con saltos pequeños lo cual implica una variación mínima en las variaciones de velocidad de los rotores. De igual forma el controlador difuso sigue atenuando de un mejor modo estas variaciones.

Respuesta a trayectoria

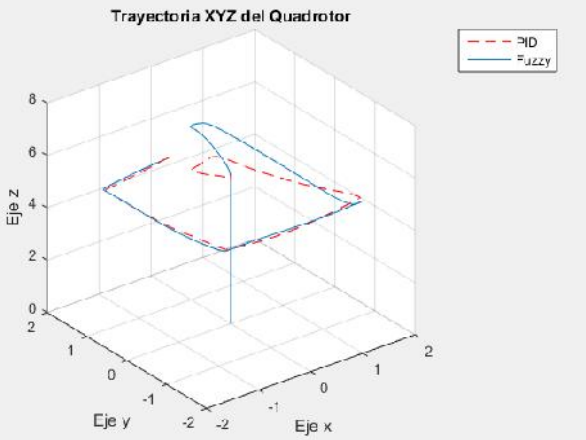
Queriéndose realizar un seguimiento a una trayectoria con forma cuadrada en los ejes XY y que tenga una altura de $5m$ el resultado del modelo completo es el mostrado en las figuras 86 - 87

Figura 86: Respuesta de seguimiento a trayectoria (ejes XY)



Fuente: Propia del autor

Figura 87: Respuesta de seguimiento a trayectoria (ejes XYZ)



Fuente: Propia del autor

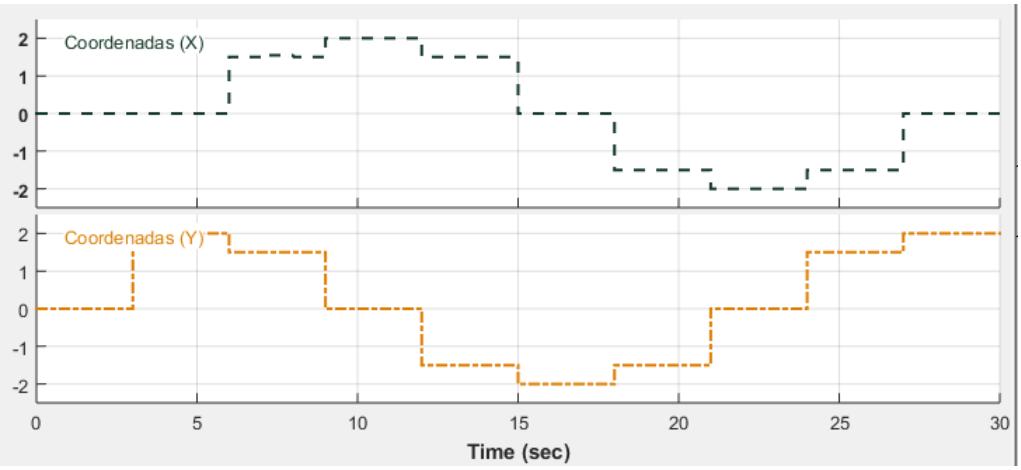
Como se observa en las figuras 86 - 87 el punto inicial del quadrotor son las coordenadas [0,0] dejando unos segundos para que el vehículo tuviese tiempo para alzar vuelo y estabilizar en su punto de referencia, posteriormente comienza a realizar la ruta establecida, mostrando una respuesta similar a la trayectoria deseada.

5.2. Segunda Prueba (Trayectoria Hexagonal)

En esta prueba se analizarán los mismos valores mencionados en la anterior prueba. Dejándose el mismo valor en la altura establecida. Se cambiarán los valores

determinados por los generadores de señales, los cuales se observan en la figura 88. El quadrotor tendrá las mismas coordenadas iniciales y valores en cero de cada componente, su tiempo de despegue seguirá siendo den 3 segundos para conseguir la altura requerida.

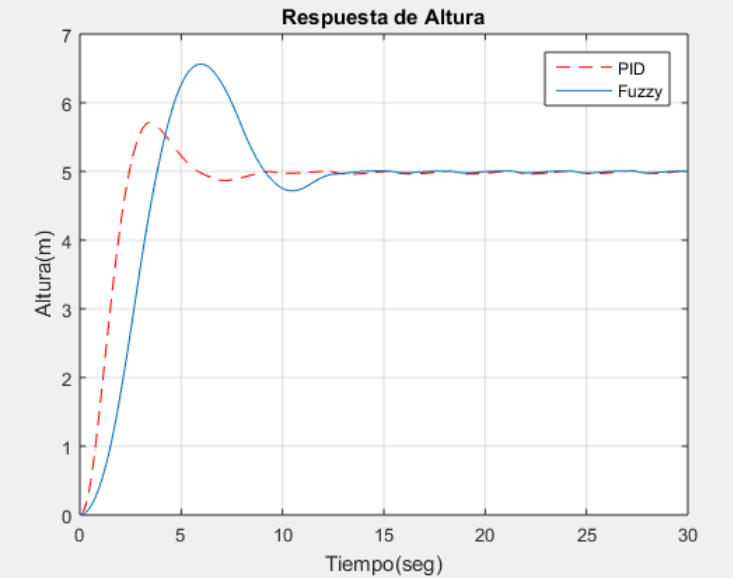
Figura 88: Generador de señales (Trayectoria Hexagonal)



Fuente: Propia del autor

Controlador de Altura

Figura 89: Comportamiento de altura.

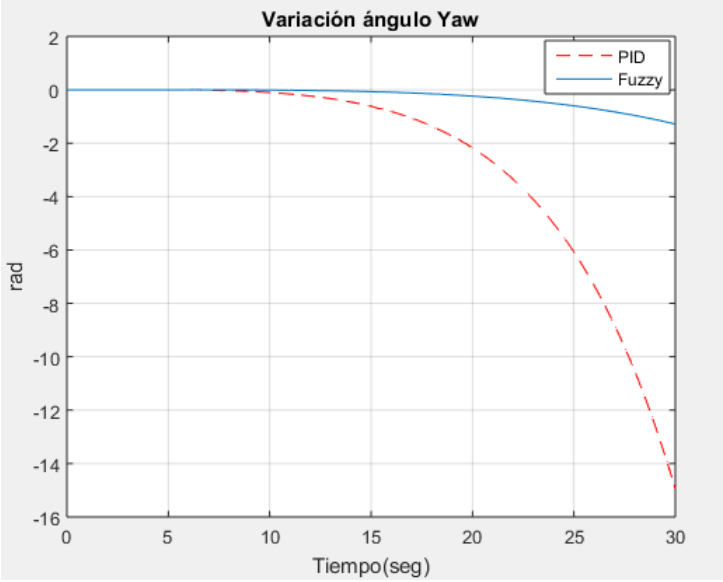


Fuente: Propia del autor

En la figura 89 se evidencia que el control PID a comparación del difuso es más rápido al momento de estabilizar el sistema, dando un tiempo de establecimiento de 10 segundos para el control PID y uno de 15 segundos para el difuso. También como se mencionó en la primera prueba el sobre impulso del controlador difuso es mayor con un 35 % mientras que el PID tiene un 17 %.

Controlador de Yaw

Figura 90: Comportamiento de Altura.

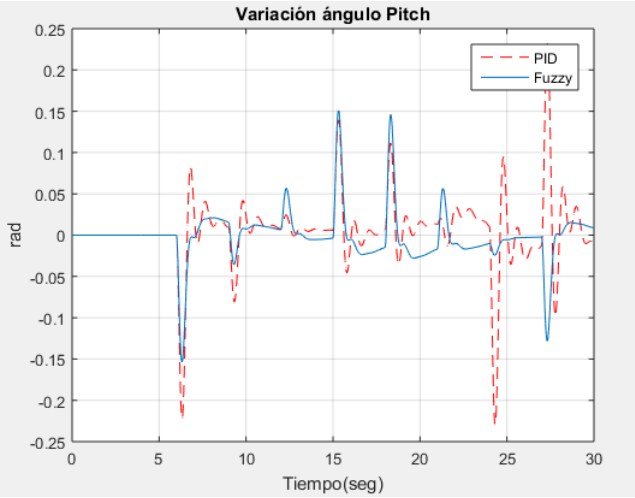


Fuente: Propia del autor

En la figura 92 se puede ver que tiene el mismo comportamiento exponencial que la figura 83. De lo cual se deduce que el control difuso es estabiliza más los rangos de giro en la variable Yaw dado que su variación en sentido horario es de aproximadamente 1 rad lo cual daría un rango de giro de $57°$, por otro lado, el control PID no es muy confiable dado que su valor es de 15 rad lo cual implica 2.5 vueltas en su mismo eje.

Controlador de Pitch

Figura 91: Comportamiento de Pitch

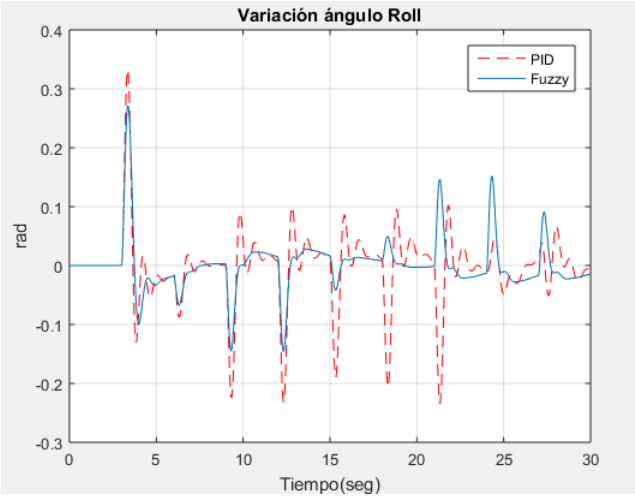


Fuente: Propia del autor

En la figura 91 se evidencia el mismo comportamiento mostrado en la figura 84, dejando ver un comportamiento no uniforme, pero con una mejor atenuación por parte del controlador difuso con respecto al controlador PID. Los rangos de giro máximos para cada control son de $13°$ para el controlador PID y de $8{,}5°$ para el controlador difuso.

Controlador de Roll

Figura 92: Comportamiento de Roll

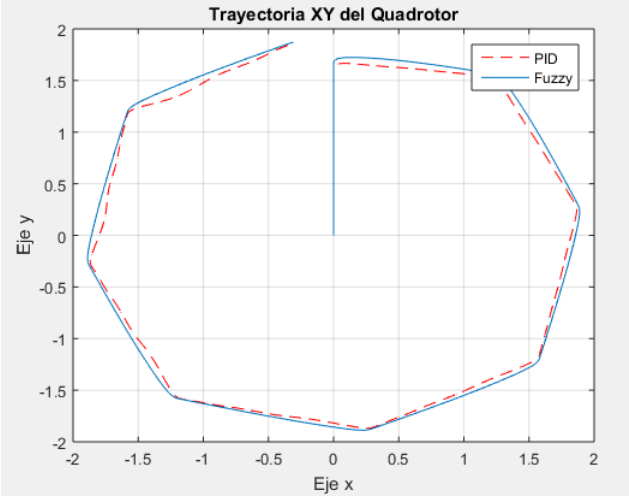


Fuente: Propia del autor

El comportamiento de Roll es similar al de Pitch el cual tiene picos en la señal producidos por la trayectoria, pero con saltos pequeños lo cual implica una variación mínima en los cambios de velocidad en los rotores.

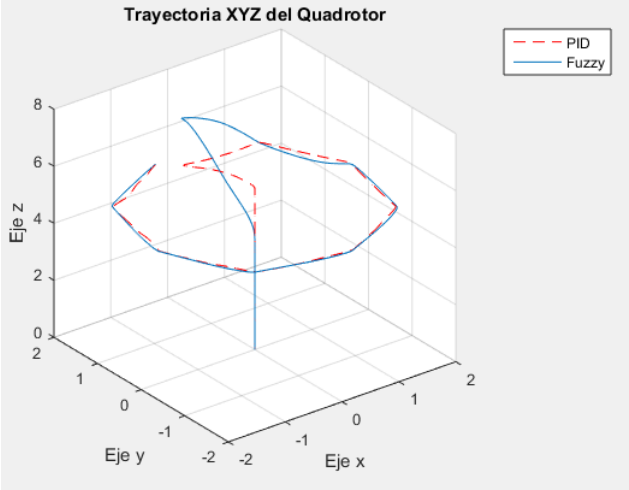
Respuesta a trayectoria

Figura 93: Respuesta de seguimiento a trayectoria (ejes XY)



Fuente: Propia del autor

Figura 94: Respuesta de seguimiento a trayectoria (ejes XYZ)



Fuente: Propia del autor

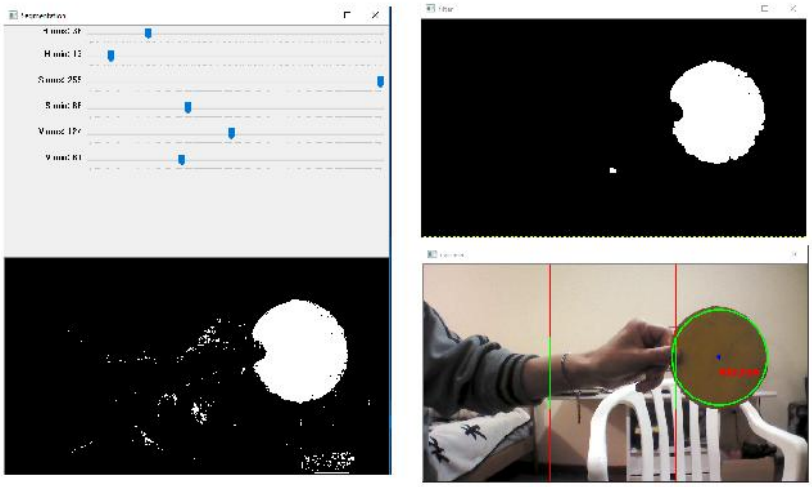
En la figura 93 se observa la respuesta satisfactoria del seguimiento de la tra-

yectoria de ambos controladores, dejando ver un mejor comportamiento por parte del controlador difuso. Con respecto al seguimiento de trayectoria en los tres ejes visto en la figura 94, se observa un mejor desempeño global por parte del controlador PID, debido a que su asentamiento en la altura se asemeja a la trayectoria propuesta.

5.3. Resultado algoritmo procesamiento de imágenes en serie

Como se mencionó en la sección 4.2 el algoritmo en serie detecta el objeto por su color. La figura 95 muestra el resultado del algoritmo el cual tiene un resultado satisfactorio en la detección del objeto. También se puede observar las coordenadas del centroide del objeto y el ruido eliminado por las operaciones morfológicas.

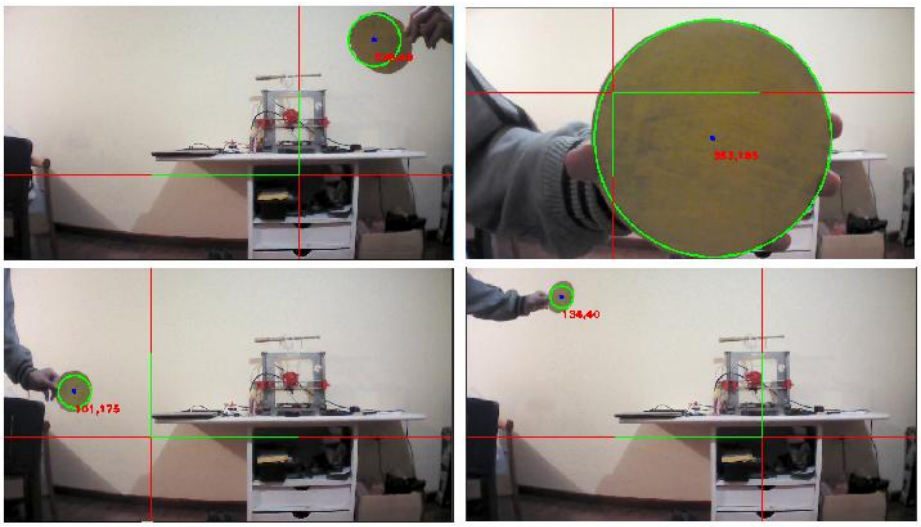
Figura 95: Detección objeto



Fuente: Propia del autor

Por otro lado, la figura 96 muestra la precisión de la detección del objeto cuando se encuentra cerca o lejos que igual es satisfactoria. Para este algoritmo se recomiendo utilizar un color que no es común en un ambiente exterior para no tener error en la detección.

Figura 96: Detección objeto



Fuente: Propia del autor

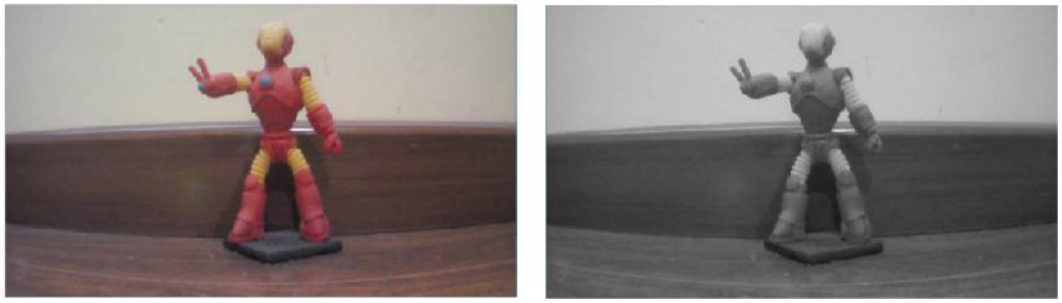
5.4. Resultados algoritmos procesamiento de imágenes en paralelo

En la siguiente sección se muestra los resultados y tiempo de ejecución de los algoritmos en paralelo. Para el tiempo de procesamiento se utiliza imagenes de tamaño 256x256, 512x512 y 1280x720, para los tiempos no se tuvo en cuenta el tiempo de transferencia de datos.

5.4.1. Conversión RGB a escala de grises

La figura 97 muestra el resultado de la conversión RGB a escala de grises y la tabla 5 el tiempo de procesamiento.

Figura 97: Conversión escala de grises GPU



Fuente: Propia del autor

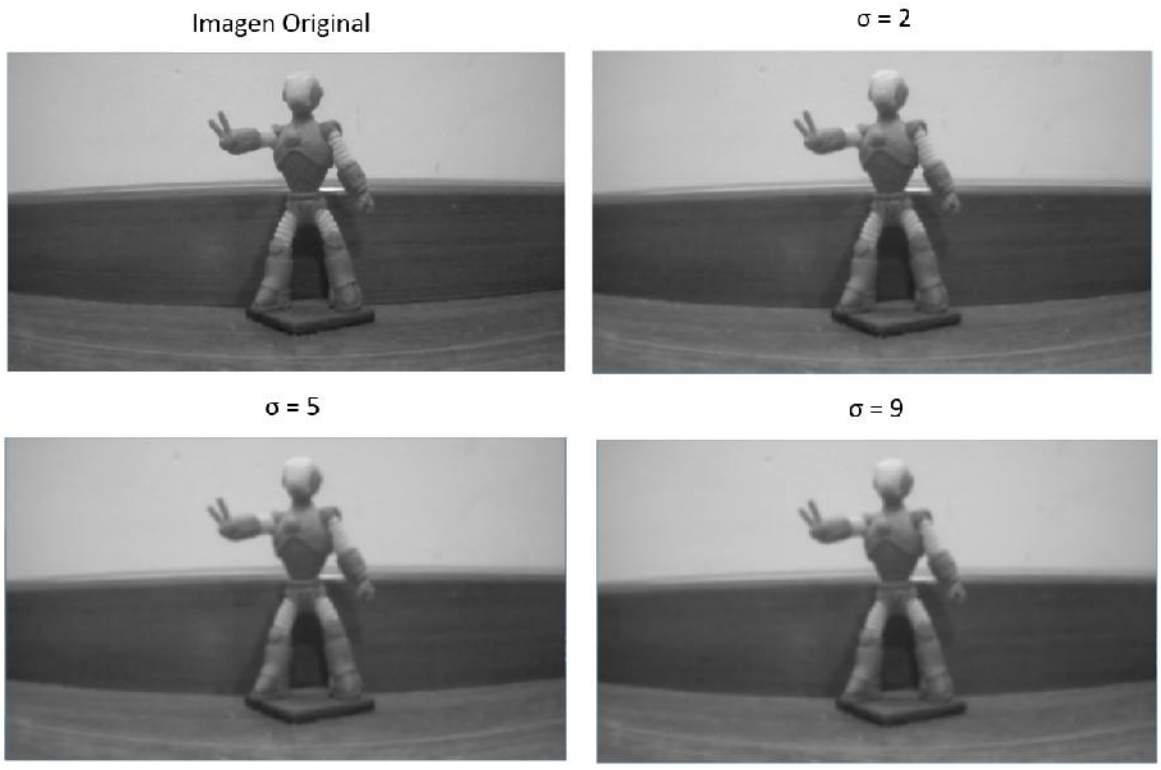
Tabla 5: Tiempo algoritmo RGB a escala de grises

Resolución Imagen	CPU(ms)	GPU(ms)
256x256	216.01	0.0294
512x512	245.21	0.0319
1280x720	251.98	0.0703

5.4.2. Filtro Gaussiano

La figura 98 muestra el resultado del filtro gaussiano para un valor de σ diferente y la tabla 6 el tiempo de procesamiento. Como se puede observar al incrementar el σ se aumenta el suavizado pero la imagen pierde detalles.

Figura 98: Filtro Gaussiano GPU



Fuente: Propia del autor

Tabla 6: Tiempo algoritmo gaussiano

Resolución Imagen	CPU(ms)	GPU(ms)
256x256	1.61	1.39
512x512	5.72	5.06
1280x720	19.31	17.38

5.4.3. Filtro Sobel

La figura 99 muestra el resultado del filtro sobel y la tabla 7 el tiempo de procesamiento. El resultado del filtro es satisfactorio ya que detecta los bordes correctamente. Se hizo la comparación de este filtro con el filtro del módulo CUDA de OpenCV y se obtuvo que el filtro de OpenCV genera mucho más ruido y no detecta los bordes correctamente.

Figura 99: Filtro Sobel GPU

Fuente: Propia del autor

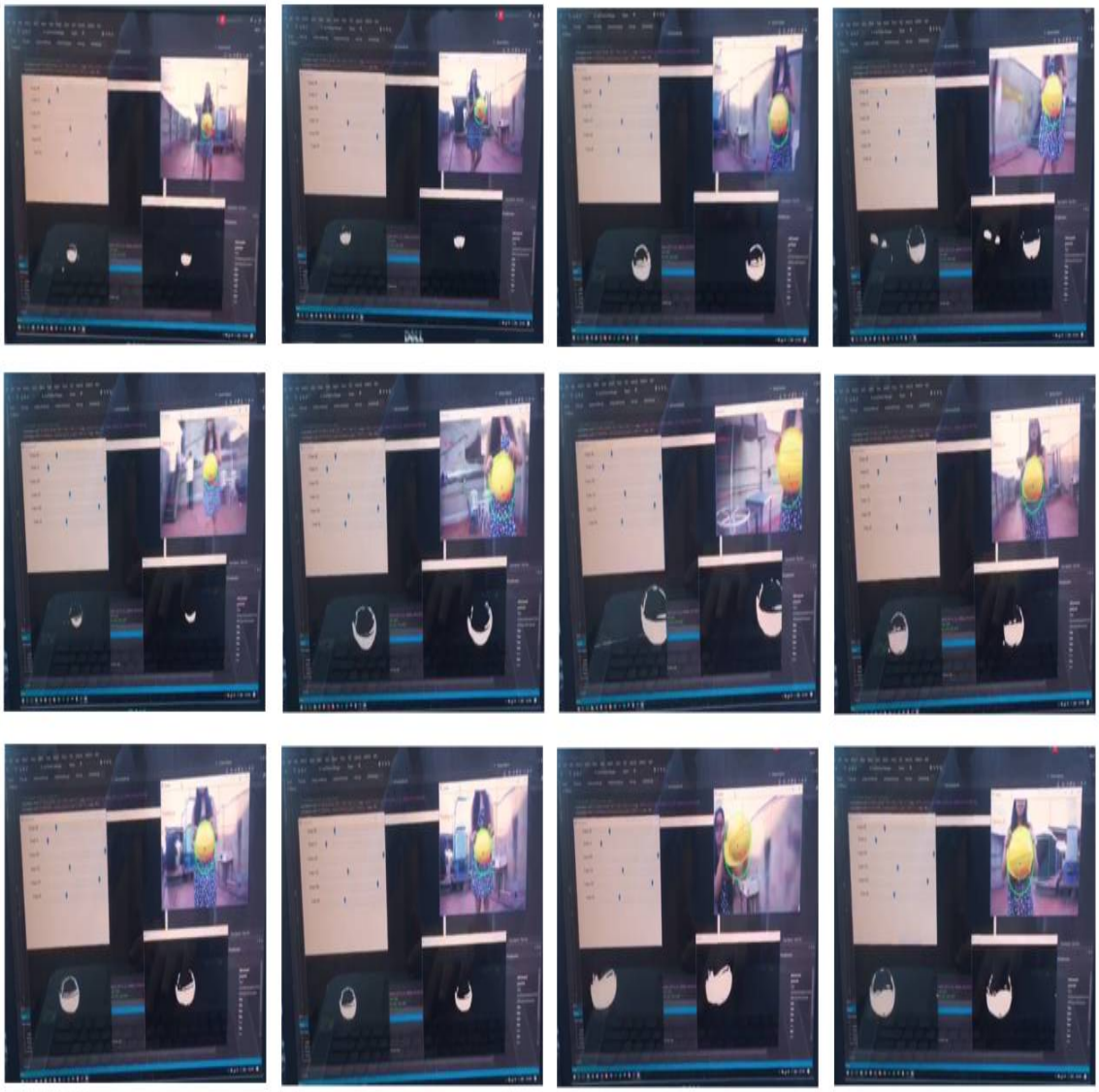
Tabla 7: Tiempo algoritmo sobel

Resolución Imagen	CPU(ms)	GPU(ms)
256x256	54.44	0.147
512x512	64.11	0.415
1280x720	74.80	1.34

5.4.4. Seguimiento del Objeto

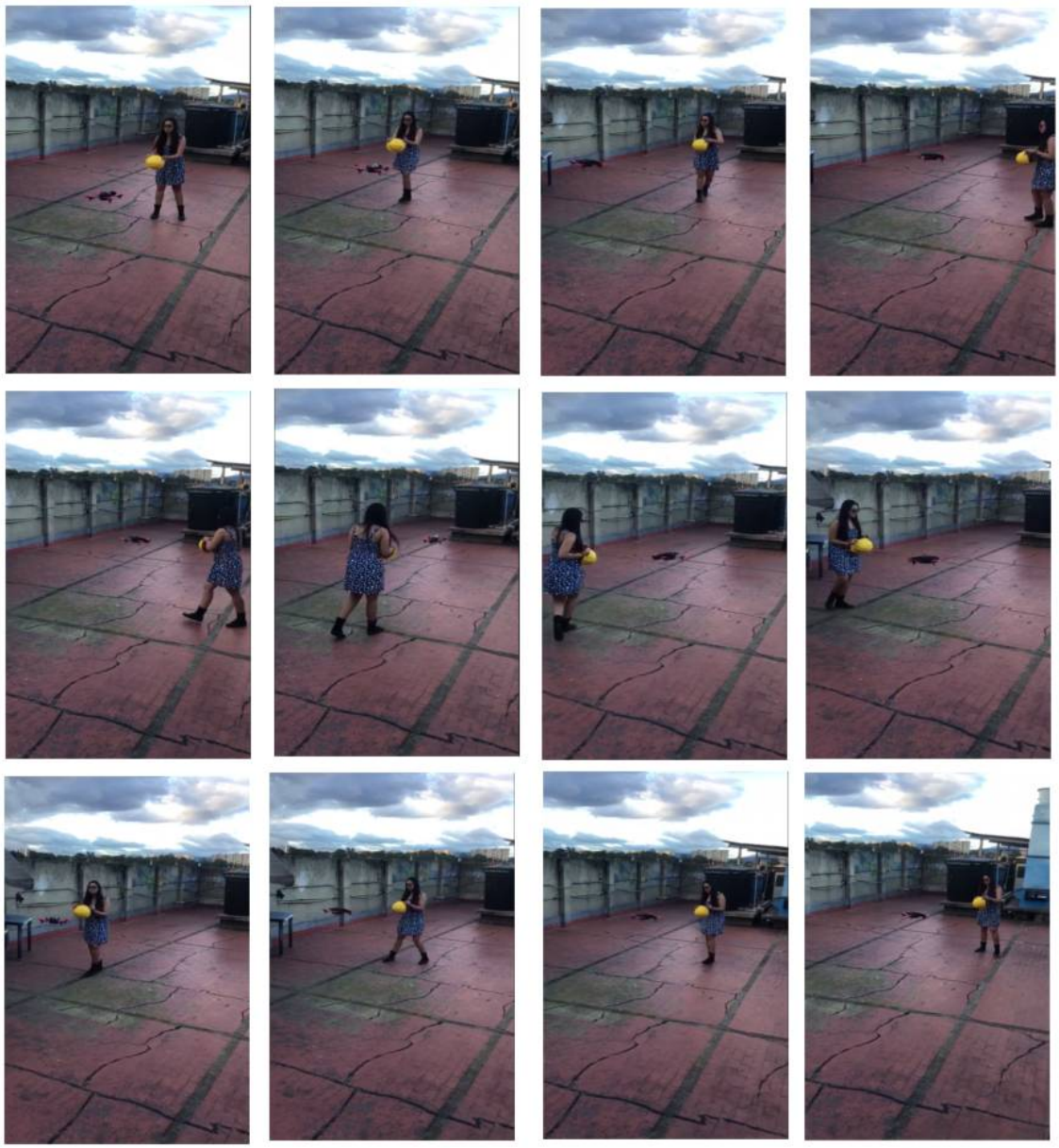
El seguimiento del objeto se hizo en un ambiente externo donde se mueve el objeto en diferentes direcciones y el AR Drone lo sigue. En la figura 100 se puede observar la secuencia de imágenes donde el Ar Drone reconoce el objeto y la figura 101 el seguimiento del objeto.

Figura 100: Detección del objeto en estación remota



Fuente: Propia del autor

Figura 101: Seguimiento objeto ambiente externo



Fuente: Propia del autor

En el enlace `https://youtu.be/HJ1br0fggl8` se puede ver el funcionamiento del Ar Drone siguiente el objeto.

6. Conclusiones y Trabajos Futuros

6.1. Conclusiones

- El objetivo principal del proyecto es desarrollar un algoritmo de procesamiento de imágenes para el reconocimiento y seguimiento de objetos en ambientes externos. El objetivo fue alcanzado con el desarrollo del algoritmo de procesamiento de imágenes utilizando la librería OpenCV en lenguaje de programación C++. Por otro lado, para el seguimiento fue desarrollado bajo reglas clásicas de control como lo son el PID y la lógica difusa.

- Mediante el análisis del modelo matemático de un vehículo aéreo tipo quadrotor se puede inferir que el vehículo no es lineal y costa de 6 grados de libertad, tres de ellos definen la posición de un punto de referencia en la estructura y los otros tres definen la orientación del cuerpo.

- El algoritmo de procesamiento de imágenes en serie cumple con la detección del objeto por su color. A su vez, se puede ajustar los valores de H, S, y V con los trackbars mientras el algoritmo se está ejecutando así adaptándose al ambiente. Así mismo, los algoritmos desarrollados en paralelo cumplen con los resultados esperados para ser empleados en trabajos futuros.

- La implementación de la lógica difusa en el diseño de sistemas de control es un método de gran capacidad y eficacia, ya que solo se necesita el conocimiento de un experto que conozca el comportamiento de modelo a controlar, permite realizar toma de decisiones similares a como lo haría un humano, pero con aproximaciones numéricas.

- Gracias a las satisfactorias respuestas obtenidas por los controladores PID y Fuzzy se deduce que el controlador PID obtiene un mejor rendimiento con respecto a la variable de altura, dado que sus tiempo de estabilización y sobre impulso son menores al difuso; pero en el momento de seguir la trayectoria con coordenadas en los ejes X y Y el controlador Fuzzy muestra señales con valores más reducidos en su error. En términos generales con el controlador PID se obtienen mejores respuestas al compararlo con el controlador por lógica difusa, dando así una comparación entre error del 7 % para el control PID y 10 % al control Fuzzy en relación a las referencias.

6.2. Trabajos Futuros

- Desarrollar la reconstrucción morfológica y transformada watershed en paralelo para la detección del objeto e incorporarlos con el filtro gaussiano y filtro sobel que ya fueron desarrollados en GPU. Con estos cuatro algoritmos se obtiene un sistema de visión computacional más robusto.

- Implementar el uso de la memoria compartida y la memoria de textura para un mejor rendimiento en los algoritmos. El uso de estas memorias disminuye el tiempo de transferencia de datos y disminuye el uso de la memoria global de la GPU.
- Mejorar el filtro gaussiano y cambiarlo por un filtro bilateral. El filtro bilateral sigue el mismo enfoque que el filtro gaussiano en suavizar la imagen, pero deja los bordes de la imagen, lo que mejoraría el resultado del filtro sobel.
- Ya que el diseño del controlador PID fue hecho por métodos heurísticos, dejando para trabajos futuros la implementación del control por métodos desarrollados.
- Dado el modelo se podría implementar un controlador PID adaptativo el cual permita graduar el valor de las ganancias según el comportamiento de la planta.
- Otro método de control que se puede implementar es el LQR el cual trabaja bajo las frecuencias del modelo.

Referencias

[1] PINILLA, J. W. El futuro del vuelo no convencional. Ed. Colombia Aprende. 2008. [Documento de Word]. http://www.colombiaaprende.edu.co/html/docentes/1596/article-169055.html

[2] JAIN, R. KASTURI, R. SCHUNCK, B. G. Machine Vision. 1. ed. New York: McGraw-Hill International Editions, 1995.

[3] KRANIK, Tomas. VONASEK, Vojtech. FISER, Daniel. FAIGL, Jan. AR-Drone as a Platform for Robotic Research and Education. Research and Education in Robotics: EUROBOT 2011, Heidelberg, Springer. 2011

[4] CACERES, C. HURTADO, D. RAMOS, O. Methodology for pest damage recognition in Begonia semperflorens (sugar flower) crop through image processing. Facultad de Ingeniería. Grupo GAV. Universidad Militar Nueva Granada, Bogotá, Colombia.

[5] SASKA, Martin. KRANIK, Tomas. PREUCIL, Libor. Cooperative UAV-UGV autonomous indoor surveillance. Czech Technical University in Prague

[6] HEPPNER, G. ROENNAU, A. DILLMAN, R. Enhancing sensor capabilities of walking robots through cooperative exploration with aerial robots. En. Journal of Automation, Mobile Robotics and Intelligent Systems. No. 7(2013); p. 5-11

[7] CANTELLI, L. MANGIAMELI, MELITA, C.D. MUSCATO, G. UAV/UGV cooperation for surveying operations in humanitarian demining. Università degli Studi di Catania. Dipartimento di Ingegneria Elettrica Elettronica e Informatica

[8] SILVA, A. MENDOZA, L. PEÑA, C. Inspection and monitoring system using an aerial robot guided by artificial vision. En. ITECKNE No. 10(2013); p. 190 - 198

[9] BERNARDES, Giovani. Rastreamento de alvo móvel em mono-visão aplicado no sistema de navegação autônoma utilizando GPU. Campinas, 2010. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica.

[10] BROWN, Dane. CONNAN, James. GHAZIASGAR, Mehrdad. Faster Upper Body Pose Estimation Using CUDA. University of Western Cape, Rhodes University. Department of Computer Science.

[11] YONG, Cao. PARK, Seung In. PONCE, Sean. HUANG, Jang. QUEK, Francis. Low-Cost, High-Speed Computer Vision using NVIDIA CUDA Architecture. Blacksburg. Virginia Polytechnic Institute and University. Center of Human Computer Interaction.

[12] ZHANG, Haiyang. MARTIN, Fred. CUDA Accelerated Robot Localization and Mapping. Lowell. University of Massachusetts. Computer Science Department.

[13] SHINSEL, Amber. Implementing Laser Scanning for Robotics Vision with CUDA. 2010.

[14] RUD, Maxim. PANTIYKCHIN, Alexander. Development of GPU – accelerated localization system for autonomous mobile robot. Tomsk, 2014. National Research Polytechnic University. International Conference on Mechanical Engineering, Autornation and Control Systerns

[15] VLADIMIR, Tyan. DOO-HYUN, Kim. YOUNG-GUK, Ha. DONGWOON, Jeon. Fast Multi-Line Detection and Tracking with CUDA for Vision-Based UAV Autopilot. Seoul, 2014. Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing

[16] SANCHEZ, Sergio. Diseño e implementación de una cadena completa para desmezclado de imágenes hiperespectrales en tarjetas gráficas programables (GPUs) [online]. UNIVERSIDAD DE EXTREMADURA.

[17] BOUABDALLAH, Samir. MURRIERI, Pierpaolo. SIEQWART, Roland. Design and control of an indoor micro quadrotor. 2004.

[18] BOUABDALLAH, Samir. Design and control of quadrotors with application to autonomous flying. 2007. Ecole Polytechnique Federale de Lausanne.

[19] FORSHAW, Jason. Lappas, Vaios. Architecture and systems design of a reusable martian twin rotor tailsitter. Acta Astronautica, vol. 80, pp. 166-180, 2012.

[20] OLFATI-SABER, Reza. Nonlinear control of underactuated mechanical systems with application to robotics and aerospace vehicles. PhD thesis, Massachusetts Institute of Technology, 2001.

[21] RAFFO, Guilherme. Modelado y control de un helicoptero quadrotor. Dept. telematica, automatica y robotica, Univ. de Sevilla, 2007.

[22] MISTLER, V. BENALLEQUE, A. MSIRDI, N. Linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback,ïn Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on, pp. 586-593, IEEE, 2001.

[23] CASTILLO, Pedro. LOZANO, Rogelio. DZUL, Alejandro. Modelling and control of mini-flying machines. Physica-Verlag, 2005

[24] SEBE, Nicu. LEW, Michael. Robust Computer Vision: Theory and Applications[online]. Leiden Institute of Advanced Computer Science.

[25] FORSYTH, David. PONCE, Jean. Computer Vision: A Modern Appoach. 2003. Pearson. ISBN13: 9780130851987

[26] SHAPIRO, Linda. STOCKMAN, George. Computer Vision[online]. University of Washington, Michigan State University.

[27] PRISCO, Giulio. TSLA Stock: NVIDIA Powered Computer Vision System Will Drive Tesla Motors Inc[online].

[28] BARNGROVER, Christopher. Computer Vision Techniques for Underwater Navigation. Univerity of California, San Diego.

[29] JACK, Keith. Video Demystified: A Handbook for the Digital Engineer, 5th Edition. 2007. Newnes. ISBN13: 9780750683951

[30] ACHARYA, Tinku. RAY, Ajoy. Image Processing: Principles and Applications. 2005. Wiley. ISBN: 978-0-471-71998-4

[31] SZELISKI, Computer Vision Algorithms and Applications. 2011. Springer. ISBN 978-1-84882-935-0

[32] BRADSKI, Gary. KAEHLER, Adrian. Learning OpenCV. 2008. ISBN: 978-0-596-51613-0.

[33] FLUSSER, Jan. ZITOVA, Barbara. SUK, Tomas. Moments and Moment Invariants in Pattern Recognition. 2009. Wiley. ISBN: 978-0-470-69987-4

[34] PACHECO, Peter. An Introduction to Parallel Programming. 2011. Morgan Kaufmann. ISBN-13: 978-0123742605

[35] CHALERMWAT, Prachya. ALEXANDRIDIS, Nikitas. PIAMSA-NGA, Punpiti. O'CONNELL, Malachy. PARALLEL IMAGE PROCESSING IN HETEROGENEOUS COMPUTING NETWORK SYSTEMS[online]. The George Washington University. Disponible en:

[36] SARABIA, Martin. Paralelizacion de filtros de correlacion para deteccion de objetos con matlab[online]. Universidad de Colima.

[37] COOK, Shane. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of Gpu Computing). 2012. Morgan Kaufmann. ISBN13: 978-0124159334

[38] INAM, Rafia. An Introdcution to GPGPU Programming - CUDA Arquitecture[online]. Mälardalen University.

[39] ULLOA, Ariel. PÉREZ, Luis. ACELERACIÓN DE ALGORITMOS CON TECNOLOGÍAS DE MULTIPROCESAMIENTO[online]. UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.

[40] CHENG, John. GROSSMAN, Max. McKERCHER, Ty. Professional CUDA C Programming. 2014. Wrox. ISBN13: 978-1118739327

[41] SETOAIN, Javier. PROCESAMIENTO DE IMÁGENES HIPERESPECTRALES EN GPUs [online]. UNIVERSIDAD COMPLUTENSE DE MADRID.

[42] AGUILAR, Pablo. RECONOCIMIENTO DE BORDES EN IMÁGENES APLICADO A ANILLOS DE ARBOLES[online]. UNIVERSIDAD DE CHILE.

[43] JEN, Jia Jun. Image Processing with CUDA[online]. University of Nevada.

[44] RAY, Debosmit. Edge detecion in Digital Image Processing. 2013. University of Washington.

[45] GONZALEZ, Rafael. Woods, Richard. Digital Image Processing. 2007. Pearson. ISBN-13: 978-0131687288

[46] GONZALEZ, Rafael. WOODS, Richard. EDDINS, Steven. Digital Image Processing Using Matlab. 2010. MHE. ISBN-13: 978-0070702622

[47] VINCENT, Luc. Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms. April 1993. Published in the IEEE Transactions on Image Processing, Vol. 2, No. 2

[48] KORBES, Andre. Análise de Algoritmos da Transformada Watershed. 2010. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e Computação

[49] ZADEH, Lofty. Fuzzy Logics in Biomedical Systems. 1991.

[50] OUSSALAH, Mourad. NGUYEN, Hung. KREINOVICH, Vladik. A new derivation of centroid defuzzication. 2001.

[51] BONIFACIO, Mart. SANZ, Alfredo. R. Neuranales y Sistemas Borrosos. 2001.

[52] ÑECO, R. REINOSO, O. GARCIA N. ARACIL R. Apuntes de Sistemas de control. ECU 2003.

[53] OGATA, K. Ingeniería de control moderna. Pearson Educación, 2003.

[54] PISKORSKI, Stephane. BRULEZ, Nicolas. ELINE, Pierre. D'HAEYER, Frederic. AR.Drone Developer Guide SDK 2.0. 2012.

[55] VAN FLEET, Patrick. Discrete Wavelet Transformations: An Elementary Approach with Applications. John Wiley and Sons, 2011. ISBN 1118030664, 9781118030660

[56] CORKE, Peter. Robotics, vision and control: fundamental algorithms in MATLAB, vol. 73. Springer, 2011.